A NETWORK FLOW APPROACH TO THE INITIAL SKILLS TRAINING SCHEDULING PROBLEM

THESIS

Anthony A. Illig, Second Lieutenant, USAF

AFIT/GOR/ENS/07-01

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this Graduate Research Project are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

A NETWORK FLOW APPROACH TO THE INITIAL SKILLS TRAINING SCHEDULING PROBLEM

THESIS

Presented to the Faculty

Department of Operational Science

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Anthony A. Illig

Second Lieutenant, USAF

December 2007

A NETWORK FLOW APPROACH TO THE INITIAL SKILLS TRAINING SCHEDULING PROBLEM

Anthony A. Illig
Second Lieutenant, USAF

Approved:

_____	_____
Roesener, August G., Capt AFIT/ENS, USAF (Advisor)	Date
_____	_____
Knighton, Shane A., Major AFIT/ENS, USAF (Reader)	Date

Abstract

The United States Air Force commissions new officers as they complete their undergraduate degree or their officer commissioning training. These officers are commissioned frequently throughout the calendar year, sometimes in large groups. In order to perform their job to the best of their abilities, they require proper training. With this in mind, it seems only natural that there exists a mathematical, repeatable and measurable method for scheduling these officers into their training courses to have them fully trained and available for Air Force duties in the timeliest manner.

This thesis demonstrates that the goal of efficiently and effectively scheduling officers into their training courses is not being met, and it provides a method which can be utilized by the Air Force to ensure that the goal is actually achieved. The formulation is based upon a minimum cost network flow problem used to perform the scheduling. The algorithm solves the problem in a polynomial amount of time (in terms of the problem size) and gives the user measurable and comparable results. Further, numerous problem instances are presented in order to explore different decision alternatives which will enable the Air Force senior leadership to make informed decisions.

Acknowledgements

First, I would like to that God. Everything we do is for His glory.

Next, I would like to thank my advisor for his hard work and dedication. His unending guidance and assistance was paramount to this endeavor.

Also, I would like to thank my wife for her support and understanding. She put up with late nights and odd hours for 15 months.

Finally, I would like to thank my parents (all 6 of them). Without you, none of this would have been possible.

Table of Contents

Page

Abstract iv

Acknowledgements v

Table of Contents vi

List of Figures viii

List of Tables ix

I. Introduction 1

- Background 1
- Necessity of the Analysis 2
- Career Field Descriptions 3
  - Pilots 3
  - Combat Systems Officers 4
  - Air Battle Managers 4
  - Intelligence Officers 5
  - Space and Missile Officers 5
  - Airfield Operations Officer 6
- Problem Statement 7
- Research Objectives 7

II. Literature Review 8

- Overview 8
- Totally Unimodular 8
- Personnel Scheduling Problem 9
  - Demand Modeling 10
  - Shift Scheduling 11
  - Line of Work Construction 11
  - Task and Shift Assignments 11
  - Solution Methods 12
- Fast Food Scheduling Using Network Flow 13
  - Mixed Integer Linear Program Formulation 13
  - Conversion to Network Flow 15
  - Conclusions 15
- Computer Lab Staffing 16
  - Phase One 16
  - Phase Two 17
  - Formulation 17
  - Application 20
  - Conclusion 21

Airline Based Scheduling Problems 21
Large Scale Crew Scheduling Problem 21
Airline Cockpit Crew Scheduling Problem 22
Summary 23

III. Methodology 24

Overview 24
Mathematical Formulation of the Current Methodology 25
Discussion of the Current Solution Methodology 27
Mathematical Formulation of the Network Flow ISTSP Algorithm 28
Discussion of the NFISTSP Formulation 32
Total Unimodularity 33
Software Programming 37
Pseudo-Code of the Program 38
Summary 39

IV. Results 40

Overview 40
Required Inputs 40
Generating Data Sets 40
Parameter Inputs 41
Outputs 44
Example Output 48
Actual Output Discussion 52
MIN TDD Versus DIST DD 52
Summary 61

V. Conclusions 62

Overview 62
Problem Specific Recommendations 62
Problems to Overcome 63
Additional Robustness 64
Future Research 64

Appendix A: SAS Code 66

Appendix B: Output Table 101

Bibliography 103

List of Figures

Page

Figure 1: Network Formulation of Phase 2 of the Scheduling Problem (Knighton 2005) 18

Figure 2: General Network Flow Formulation 29

Figure 3: Class Node Breakdown 30

Figure 4: Example Matrix 34

Figure 5: Example Class View comparing SOC Blend 48

Figure 6: Example Class View comparing AFSC Blend 49

Figure 7: MIN TDD versus DIST DD by Average Days Ineffective 52

Figure 8: Maximum Number of Down Days for an Individual 53

Figure 9: Average Ineffective Days versus ROTC Delay 54

Figure 10: Average Days Ineffective versus USAFA Leave 56

Figure 11: Effect of Blend Levels 58

Figure 12: Example ASBC Course Composition by SOC With 75% Blend 59

Figure 13: Example ASBC Course Composition by AFSC with 75% Blend 59

Figure 14: Example ASBC Course Composition by SOC With No Forced Blend 60

Figure 15: Example ASBC Course Composition by AFSC With No Forced Blend 60

List of Tables

Page

Table 1: Number of surveyed papers using each methodology for personnel scheduling (Ernst et. al., 2004b)..... 13

Table 2: Sample Output 50

A NETWORK FLOW APPROACH TO THE INITIAL SKILLS TRAINING SCHEDULING PROBLEM

I. Introduction

Background

Officers of the United States Air Force (USAF) typically are commissioned from one of three sources of commissioning (SOC): the United States Air Force Academy (USAFA), the Reserve Officer Training Corp (ROTC) or the Officer Training School (OTS). After commissioning, officers are assigned to a specific career field and designated with an Air Force Specialty Code (AFSC) associated with the career field. Each career field has a sequence of courses, called initial skills training (IST) courses, which a newly commissioned officer must complete to ensure basic competencies in the career field. While officers may be allowed to change career fields (also changing their AFSC) at some point in their Air Force (AF) career, this does not usually occur until the officer has progressed into the ranks of Captain or Major. This research has labeled the problem of assigning newly commissioned officers to their respective IST the Initial Skills Training Scheduling Problem (ISTSP). The size and scope of the ISTSP depends upon the number of career fields included in the scheduling process and the number of newly commissioned officers requiring scheduling.

The first course in the IST course progression for most USAF career fields is the Air and Space Basic Course (ASBC). On September 12th, 1997, ASBC was created under the command of the Air University (AETC 1999). One mission of ASBC is "to inspire new USAF officers to comprehend their roles as Airmen" (ASBC-CC 2006). In essence, ASBC serves to help newly commissioned officers understand the contribution of each AFSC to the overall success of the USAF.

Another goal of ASBC is to facilitate building of relationships with officers from other commissioning sources. Most officers have detailed knowledge regarding their own commissioning source, but are lacking in knowledge and understanding of the other sources. By building relationships with officers from another commissioning source (and perhaps with a differing AFSC), their knowledge of the entire Air Force expands. Ten classes of ASBC are typically taught each year at Maxwell Air Force Base (AFB) in Montgomery, AL. According to Oliver (2007), the capacity in each course differs throughout the year, but the average is approximately 392. Due to a recent mandate by the Air Force Chief of Staff, all officers must attend one of these courses in their first two years of active military duty.

ASBC is not the conclusion of IST for most officers in the Air Force; for six specific career fields, ASBC is only the beginning of the course sequence. For Air Battle Managers (ABMs), Intelligence Officers, Space and Missile Officers, Airfield Operations Officers, Pilots and Combat Systems Officers, ASBC only begins their extensive training process. Since their training covers many months and sometimes years, officers in these career fields must devote the required 6 weeks necessary to attend ASBC prior to the remaining IST courses.

Necessity of the Analysis

The current methodology no longer meets the new (and constantly changing) requirements imposed by senior leadership. It only searches for a feasible schedule without attempting to optimize any particular goal (for example, the total number of ineffective days for all scheduled officers) or achieve any blending of AFSC and SOC in a given class. By not attempting to optimize the total ineffective days, the current schedule generating methodology is costing the Unties States Air Force millions of dollars by paying salaries to unproductive officers.

In the current situations, new lieutenants are forced to wait for classes to becoming available. This time spent waiting is termed "casual status." During this time, the lieutenants are assigned menial tasks that, if they were not there, would have been accomplished by members of a unit as additional duties. In many cases, these tasks are so minor that the lieutenants consider the casual status time a "paid vacation." These "paid vacations" account for a large amount of wasted money for the Air Force.

The following discussion demonstrates the magnitude of the salaries being paid to unproductive officers. A lieutenant that is waiting for a class available (i.e. on casual status) still receives his/her salary of approximately $124 per day. This value is determined by combining the average Basic Allowance for Housing, Basic Allowance for Sustenance and Base Pay for a 2nd lieutenant in a given month. This monthly income is then multiplied by 12 to find the total annual income, and the annual income is divided by 365 to get the average daily salary. Approximately 4000 officers are scheduled on an annual basis. If the average wait time for a person is reduced by just 1 day, an approximate savings of $496,000 could be realized. Since the current method does not search for that minimum cost (in terms of number of unproductive days), this research has the potential to save the Air Force a great deal of money in salaries paid to unproductive officers.

Career Field Descriptions

Pilots

In addition to ASBC, officers wishing to become pilots must also complete Introductory Flight Screening (IFS) or Introductory Flight Training (IFT) (both terms refer to the same course) and Undergraduate Pilot Training (UPT). The original IFS program, located in Colorado Springs, CO, was terminated in October 1999 (AETC 1999). In February 2007, a new IFS program was started in Pueblo, CO (Diamond

2007). The course is meant to expose students to flying and prepare them for UPT. The course consists of classroom instruction and flight training in the Diamond DA-20 aircraft (Alcock 2006). Each IFS class can accommodate 97 students. Currently IFS is a 6 week course with 14 classes offered each year. It is possible that IFS will be condensed to a 5 week course with up to 20 classes offered each year.

UPT consists of three separate phases. In phase one, students fly Cessna T-37Bs (primary training aircraft). In phase two, students fly Raytheon T-6As (primary training aircraft). If students are selected for the fighter/bomber aircraft track, they will transition to Northrop T-38Cs (advanced trainer for fighter/bomber pilots) in phase three; otherwise, they fly Raytheon (Beech) T-1As (advanced trainer for cargo/tanker pilots) in preparation for the cargo airlifter track. All three phases are completed sequentially at the same base. It takes approximately 55 weeks to complete all three courses. UPT is taught at Columbus AFB in Columbus, MS; Laughlin AFB in Del Rio, TX; Vance AFB in Enid, OH; and Sheppard AFB in Wichita Falls, TX. While each base has a different capacity per class, the UPT classes are scheduled to simultaneously start and graduate at each of the different training bases. The total capacity of all classes at each of the bases allows for an average of 74 students to be in training at one time. Each base currently offers 15 classes per year.

Combat Systems Officers

Combat Systems Officers (CSOs) must attend Primary Course (PRIM) located at either Naval Air Station (NAS) Pensacola in Pensacola, FL or Randolph AFB, TX. This course is 27 weeks long and is offered 50 times each year (between both bases). Each of the classes at NAS Pensacola has a maximum capacity of 2 students while the classes at Randolph AFB have a capacity around 20.

Air Battle Managers

ABMs must attend the Undergraduate Air Battle Manager Course (UABMC) located at Tyndall AFB in Panama City, FL. In this course, students gain valuable knowledge ranging from radar theory to large force employment in preparation for their careers as ABMs (Tyndall 1999). The course is 32 weeks long and there are 17 classes offered each year with each class having a capacity of 9.

Intelligence Officers

Intelligence officers must attend Intelligence Officer Training (IOT) located at Goodfellow AFB in San Angelo, TX. In this course, students are trained "in collection, analysis, and application of all-source intelligence to plan and execute combat operations" (Goodfellow 1999). The course is 141 training days long and there are between 13 and 17 classes offered each year with each class having a capacity between 1 and 17 students. These numbers change each year based on the accession targets provided by Air Staff and the Non-Rated Line Officer Accessions Conference (NRLOAC). In order for a student to attend the course, they must have Top Secret (TS) security clearance.

Space and Missile Officers

Officers entering the Space and Missile career field must attend the Space 100 (S100) course located at Vandenberg AFB in Lompoc, CA. The course is 35 academic days long (5 weeks). There are between 8 and 9 classes offered each year, and each class has a capacity between 2 and 42 students. These numbers are determined by Air Staff and the NRLOAC. In order for a student to attend the course, they must have a TS security clearance. Following the course, each officer is given one of five specialties, each of which requires additional training.

The five specialty IST courses are also located at Vandenberg AFB and immediately follow the S100 courses. The five specialties are: Satellite Command and

Control Officer (SCCO), Spacelift Officer (SLO), Missile Combat Crew Officer (MCCO), Space Surveillance Officer (SSO), and Space Warning Officer (SWO).

SCCOs must attend Satellite Command and Control Fundamentals (SC2F). This course is approximately 13 weeks long. After this course, each SCCO attends one of four system specific courses: Satellite Mission Control Subsystem (SMCS), Global Positioning System (GPS), Command and Control System - Consolidated (CCS-C), Defense Satellite Communication System (DSCS). Each of these courses are 9 days long and are scheduled immediately after the completion of the SC2F courses.

SLOs must attend the 10 day long Spacelift Officer Course (SLOC) which is subsequent to completion of the S100 course. MCCOs must attend one of four Reaction Training (RT) courses. Each course is 10 weeks long and is specific for the base to which the officer will be assigned upon successful completion of the IST courses. SSOs must attend the 16 day long Active Space Surveillance (ASS) course. SWOs must attend one of four system specific courses: Phased Array Warning System, Perimeter Acquisition Radar Attack, Ballistic Missile Warning System, or Space Based Infrared Systems. The course lengths are 40 days, 23 days 32 days and 26 days respectively.

This thesis only addresses scheduling students up to the start of this series of courses. The full course description is included to show that the training course series is extensive and uninterruptable. Therefore, ASBC must start the sequence or it will not be attended.

Airfield Operations Officer

Airfield Operations Officers must attend the Airfield Operations Officer Course (AOOC) located at Keesler AFB in Biloxi, Mississippi. The course is 76 academic days long. There are between 3 and 6 classes offered each year with each class having a

capacity between 3 and 7 students. These numbers are determined by Air Staff and the NRLOAC.

Problem Statement

Air Education and Training Command (AETC), in conjunction with the Air Force Personnel Center (AFPC), has requested the development of a technique or methodology which can be used to properly schedule officers into their respective IST sequence of courses. The current methodology for generating the schedule simply seeks to find a *feasible* schedule for officers into IST courses course. It does attempt to optimize (i.e. minimize) the number of unproductive days which an officer spends on "casual" status waiting for a class to begin. By developing a methodology that seeks to minimize this cost (in terms of unproductive days), officers will spend, on average, less time to complete their IST courses and thus be able to be fully productive in their career field much sooner.

Research Objectives

The goal of this research is to produce an efficient and repeatable method for scheduling newly commissioned officers of the six critical AFSCs into their respective IST classes. The overall objective of the solution is to minimize the number of ineffective days spent waiting for training. The solution will be in a form that is easy to implement and will not require AFPC to purchase additional software or hardware.

II. Literature Review

Overview

Proper scheduling is necessary in nearly every corner of industry. For this reason, it is a popular topic in scholastic literature. It has been studied since the 1950s and remains a topic of great interest today (Ernst, et al, 2004b). As a whole, scheduling encompasses a broad array of problem instances. The problem addressed by this research can be described as a personnel scheduling problem. Numerous refereed journals and conference proceedings can be found on this specific type of scheduling problem alone.

This literature review will first define total unimodularity and explain why it is important and useful in this type of research. Next, it will discuss how to classify a personnel scheduling problem followed by examples of how personnel scheduling problems have been solved using network flow models (both with and without side constraints. The last section will outline some further research into airline based scheduling problems.

Totally Unimodular

Total unimodularity is a useful mathematical property which is exhibited by some network flow problems. By definition, a matrix **A** is said to be *totally unimodular* if and only if every square sub matrix of **A** has a determinant of -1, 0 or +1 (Bazaraa, Jarvis and Sherali 2005). Bazaraa, Jarvis and Sherali prove, by induction, that the node-arc incidence matrix that defines a network is totally unimodular. This property is also maintained when the arcs have capacity constraints. The reason this property is of importance is also proven by Bazaraa, Jarvis and Sherali (2005). They show that since **A** is totally unimodular, corner points of the feasible region defined by **A** are comprised solely of integer solutions (provided the right hand side of the constraints are integer values). Therefore, in a problem where the solution must be integers, the LP relaxation

of the problem can be solved and the resulting answer will be integer. Algorithms have been developed which can solve any LP problem while only requiring a number of mathematical operations that are polynomial in terms of the number of decision variables and constraints. This implies that all LP problems can be solved "quickly." Unfortunately, the node arc incidence matrix does not encompass all the constraints of most minimum cost network flow problems. There are often side constraints which, when included in the problem formulation, may or may not destroy the total unimodularity of a matrix.

Little research has been done in the area of determining if a matrix is totally unimodular. Two rules are presented which (individually) are necessary and sufficient to define a matrix as totally unimodular. The first is to show that the determinant of each square sub matrix is -1, 0, or 1. The second was found by Raghavachari (1976). He demonstrates that a matrix can be processed and reduced in magnitude before checking each of the square sub matrices. The downfall with both of these methods is that by checking every possible sub matrix, the problem will grow exponentially as the matrix gets larger. In the problem addressed by this thesis, the node arc incidence matrix will be very large; thus, neither of these methods will be practical. However, chapter 3 will explain certain properties of side constraints which are exhibited by the scheduling problems under consideration by this research and do not destroy the total unimodularity of the constraint matrix.

Personnel Scheduling Problem

Classification of the problem instance is the first challenge faced in any scheduling problem. Through classification, the scheduling problem can be decomposed into smaller sub problems. These sub problems are easier to comprehend and solve than the overarching problem.

It is proposed that any scheduling problem can be classified by the six sub problems it encompasses: demand modeling, days off scheduling, shift scheduling, line of work construction, task assignment, and staff assignment (Ernst, et al, 2004a).

Demand Modeling

Demand Modeling is the problem of defining the demand of personnel that is placed on a system. This is done by gathering information about the service needed, then translating it into staff requirements that will satisfy the need (Ernst, et al, 2004a). Three main techniques exist to accomplish this task.

The first is task based demand. In this case, the demand is determined by some predefined tasks. The tasks are usually defined by a time window for completion and a set of skills required. These tasks clearly outline the exact demand and how it must be met. It has been noted that "task based demand is commonly used in transport applications" (Ernst, et al, 2004a). An example of this includes creating shipping schedules.

The second method of modeling demand is through flexible demand. In this case, less is known about the demand and it must be predicted using forecasting techniques. These include (but are not limited to) queuing theory, stochastic analysis and simulation. After the demand schedule has been estimated, it can be proportioned into shifts that cover the demand. Ernst notes that "flexible demand is often needed for call centers, police services and airport ground staff" (Ernst, et al, 2004a).

The final method is shift based demand. This could be considered a simplification of task based or flexible demand. In this model, the demand is predetermined by shifts that must be covered by the staff. Nurse and ambulance scheduling offer excellent examples. In these cases, "staff levels are determined from a

need to meet service measures such as nurse/patient ratios or response times" (Ernst, et al, 2004a).

Shift Scheduling

Once the demand has been found through one of the three methods above, shifts can be made to satisfy that demand. Shift selection is defined as finding the minimum number of shifts that meet the demand of the system while meeting the constraints of the problem. "Days off scheduling" is the problem of creating a schedule that allows for breaks or appropriate time off between tasks or shifts. Often, days off scheduling is done as a part of shift scheduling.

When dealing with task based demand, the problem of shift scheduling is often referred to as crew scheduling (Ernst, et al, 2004a). For this, the problem involves grouping the tasks into feasible sets that a single crew would complete. In the case of flexible demand, the shift scheduling problem is to create shifts that meet the predicted demand while allowing for appropriate time off for the staff (Ernst, et al, 2004a).

Line of Work Construction

Once shifts are defined, they must be assigned to staff members. Line of work construction involves creating a feasible set of shifts for each staff member. In many scheduling cases, the staff consists of a heterogeneous work force. That is, they posses different skill sets, wage rates, preferences, availabilities or other discriminating traits. The shifts assigned to them must be feasible to their individual situations. There can also be constraints about the sets of tasks themselves, "for example it might not be possible to immediately follow a sequence of night shifts with a day shift" (Ernst et. al., 2004a).

In the case of flexible demand, the problem is often called tour scheduling and in the case of crew pairings, it is called crew rostering (Ernst et. al., 2004a).

Task and Shift Assignments

Task and shift assignments are considered subsections of shift scheduling and line of work construction. In scheduling problems, there is often more than one task assigned to a shift. In task assignment, personnel in a shift are scheduled to individual tasks. Shift assignment "involves assignment of individual staff to the lines of work" and "is often done during the construction of the work lines" (Ernst et. al., 2004a).

Through this problem definition, the major sub problems are presented: shift scheduling and line of work construction. In most literature, the demand modeling portion of the problem is assumed to already be accomplished or be simple to determine. This is because once the demand model has been created it does not need to be recreated at each iteration. Therefore, since the process to estimate the demand model is only accomplished once, it does not matter how long the process takes.

Solution Methods

Personnel scheduling problems have been approached from a variety of solution methods. Table 1 summarizes the solution methods found in the papers surveyed by Ernst et al (2004a). If a paper uses more than one method it is counted in more than one area. In the next two sections, previous techniques will be presented that solved these two sub problems.

Method	Papers	Method	Papers
Branch-and-Bound	14	Lagrangean Relaxation	32
Branch-and-Cut	9	Linear Programming	35
Branch-and-Price	30	Matching	36
Column Generation	48	Mathematical	27
Constraint Logic Programming	46	Iterated Randomized Construction	5
Constructive Heuristic	133	Other Meta-Heuristics	11
Dynamic Programming	17	Other Methods	35
Enumeration	13	Queuing Theory	32
Evolution	4	Set Covering	58
Expert Systems	15	Set Partitioning	72
Genetic Algorithms	28	Simple Local Search	39
Goal Programming	19	Simulated Annealing	20
Integer Programming	139	Simulation	31
Network Flow	38	Tabu Search	16

Table 1: Number of surveyed papers using each methodology for personnel scheduling (Ernst et. al., 2004b)

Fast Food Scheduling Using Network Flow

From the number of papers regarding the subject, it is clear that this problem has been well studied. The solution method proposed by this thesis is based on a network flow model. Network flow modeling has been used across industries for the crew-rostering problem and tour scheduling problem. Of particular interest to this thesis is the use of network flow modeling in solving the tour scheduling problem.

Mixed Integer Linear Program Formulation

Love and Hoey examine scheduling employees at a fast-food restaurant (1990). They consider a heterogeneous work force based on work availability, skills, necessary hours per week, and wage rates. They use a mixed integer linear program (MILP) to formulate the problem. The MILP formulation is shown below.

Variables

$b_{h,l}$ = number of employees needed to work in time period h at work area l

$x_{j,k,l}$ = number of employees to work shift j on day k in work area l

$a_{h,j,k}$ = 1 if time period h is part of shift j on day k, 0 otherwise

$s_{h,l}$ = surplus employees for time period h in work area l

$w_{i,j,k,l}$ = 1 if employee i is to work shift j on day k in work area l, 0 otherwise

$r_{j,k,l}$ = shortage of employees to work shift j on day k in work area l

$y_{i,k}$ = 1 if employee i is scheduled to work on day k, 0 otherwise

z_i = number of days employee i is scheduled to work

v_k = 1 if employee i is to work at least k days in the week, 0 otherwise.

Objective Function

$$\text{Minimize } \sum_{h,l} f_{h,l} s_{h,l} + \sum_{i,j,k,l} c_{i,j,k,l} w_{i,j,k,l} + \sum_{j,k,l} g_{j,k,l} r_{j,k,l} + \sum_{ik} d_{i,k} v_{i,k} \qquad (1)$$

Subject to

$$\sum_{j,k} a_{h,j,k} x_{j,k,l} - s_{h,l} = b_{h,l} \qquad \forall h,l \qquad (2)$$

$$\sum_{i} w_{i,j,k,l} + r_{j,k,l} - x_{j,k,l} = 0 \qquad \forall j,k,l \qquad (3)$$

$$\sum_{j,l} w_{i,j,k,l} - y_{i,k} = 0 \qquad \forall i,k \qquad (4)$$

$$\sum_{k} y_{i,k} - z_i = 0 \qquad \forall i \qquad (5)$$

$$\sum_{k} v_{i,k} - z_i = 0 \qquad \forall i \qquad (6)$$

$$y_{i,k} \le 1 \qquad \forall i,k \qquad (7)$$

$$v_{i,k} \le 1 \qquad \forall i,k \qquad (8)$$

Conversion to Network Flow

In order to solve the problem, Love and Hoey suggest splitting the problem into two sub-problems. The first problem involves determining the shifts that must be manned by the employees (the 'shift scheduling' process previously described). These shifts will minimize the number of surplus hours scheduled while still meeting the manning needs of the system. This is simply defined as minimizing

$$\sum_{h,l} f_{hl} s_{hl} \qquad (9)$$

subject to constraint (2). In constraint (2), it is apparent that all nonzero entries in any column are consecutive. This implies that the problem can be solved as a minimum cost network flow problem (because it maintains total unimodularity). The side constraints in this thesis will follow a similar form.

The second sub problem to be solved assigns employees to the shifts defined by the first sub problem (the 'line of work construction' step previously described). Mathematically, this requires minimizing the remaining portion of the objective function subject to constraints (3) through (8). Each column in equations (3) through (6) has at most two nonzero entries. Further, these equations can be ordered in such a way that each of the columns with two nonzero entries are a -1 and a +1. When this order is created, a sufficient (but not necessary) condition for total unimodularity is maintained. That is, if the following hold true, then matrix *A* is totally unimodular:

1. *A* all entries 0, 1 or -1
2. *A* has no more than 2 non-zero entries in each column
3. For each column with two non-zero entries, the sum of those entries is zero

Since this condition is met, the matrix is totally unimodular.

Conclusions

The authors note that since both problems can be solved as minimum cost network flow problems, a network simplex algorithm should be used and the algorithm will converge in polynomial time. This formulation, however, has its limitations. The authors admit that it does not do well with problems where 24-hour operations are being scheduled. Further, there are limitations on how heterogeneous the work force can be. Even with these potential limitations, their use of the network flow structure of the tour-scheduling problem is very insightful and significant in solving the problem in a reasonable amount of time.

Computer Lab Staffing

Knighton also shows the power of a network formulation in scheduling technicians in the Arizona State University (ASU) computer labs (2005). In their formulation, four computer labs must be staffed by technicians. The workforce is heterogeneous in the following ways: availability to work (differs by day of the week and time of day), shift restrictions, maximum and minimum hours of work per week, personal preferences and qualifications. All of these factors are considered in the scheduling process. In addition to the staff being different, the four labs have differing staffing requirements and operating hours.

Phase One

The solution proposed consists of two phases. The first involves shift selection (described earlier in this chapter). The scheduler can assign different weights to shift lengths in order to bias the model to create longer or shorter shifts. Because selecting the shifts for computer labs are independent of each other, four smaller phase 1 problems are solved individually. In this phase, the goal is "to select a set of shifts that covers the lab requirements, as well as minimizes the sum of 'shift penalties'" (Knighton 2005). These "shift penalties" are based on the weights the scheduler assigned to different shift lengths.

This problem has been shown to be an NP-Hard problem (Knighton 2005), which implies that no known algorithm exists which can solve every instance of the problem using a number of mathematical operations which is a polynomial function of the problem size. However, because the problem can be broken down into four smaller problems, the set-covering integer programming problem can be directly used.

Phase Two

Phase two of the problem involves assigning workers to the shifts created in phase one. This is the more difficult problem of the two under consideration and thus is where the majority of the research is directed. This phase also outlines the solution approach later proposed by this thesis. Knighton states that "the tour-scheduling problem is inherently a binary set-covering problem" (2005). A binary set-covering problem is NP-Hard because it is an optimization of the set-covering problem which is NP-Complete (Vazirani 2004). Because the binary set-covering problem is NP-Hard, there is no algorithm to solve it that executes in a polynomial amount of time. Knighton, however, "developed a formulation of the problem as a minimum cost network-flow, using arc capacity method, so that the resulting network structure many times provides integer binary answers." Because the structure is based on a network, the constraint matrix is totally unimodular (Bazaraa, Jarvis and Sherali 2005).

Formulation

A pictorial representation of the network formulation for phase two is shown in Figure 1; the mathematical formulation for the problem follows the figure.

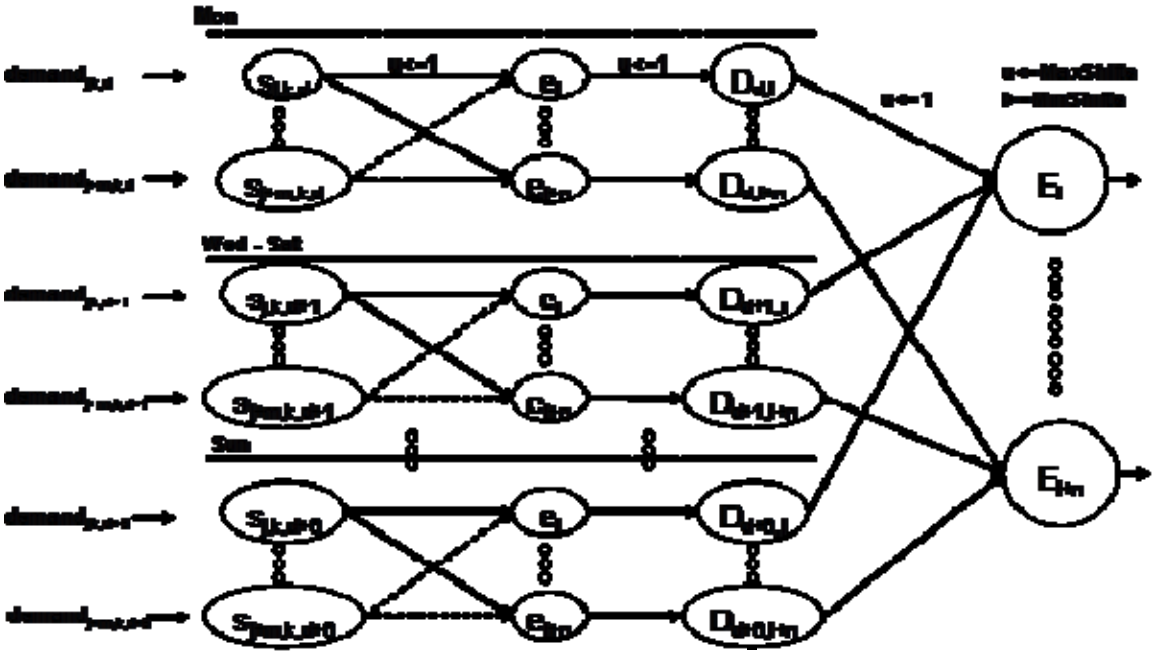

Figure 1: Network Formulation of Phase 2 of the Scheduling Problem (Knighton 2005)

Variables

$s_{j,k,d}$ = shift number j (of which there are m) requiring skill set k on day d

e_i = employee number i (of which there are n)

$D_{d,i}$ = total number of shifts on day d for employee i

E_i = total number of shifts per week for employee i

Network Flow Balance equations

$$demand_i = \sum_i FLOW(s_{j,k,d}, e_i) \qquad \forall j,k,d \qquad (10)$$

$$\sum_{j,k} FLOW(s_{j,k,d}, e_i) = FLOW(e_i, D_{d,i}) \qquad \forall i,d \qquad (11)$$

$$\sum_d FLOW(e_i, D_{d,i}) \le E_i^{(\max)} \qquad \forall i \qquad (12)$$

$$\sum_d FLOW(e_i, D_{d,i}) \ge E_i^{(\min)} \qquad \forall i \qquad (13)$$

$$FLOW(e_i, D_{d,i}) \le 1 \qquad \forall i,d \qquad (14)$$

This network flow formulation does not encompass all possible constraints for this problem; side constraints must be included. Knighton accounts for continuous 24 hour operations. They "use the appropriate industry standard for how much time is required between the end of shift *j* with skill set *k* on day *d* and the start of the shifts on day *d+1* for which employee *i* is qualified" (Knighton 2005). This leads to an array *J*, which consists of all the shifts in confliction. That is, if an employee works shift *j* which is in *J*, that employee cannot work any other shifts in *J*. Side constraint (15) is added to the formulation to account for this.

$$\sum_{j \in J} FLOW(s_{j,k,d}, e_i) \leq 1 \qquad \forall J \qquad (15)$$

When side constraints are added to a network flow formulation, the solution will sometimes be non-integer. The side constraints considered by Knighton do not always destroy the integrality. The reason behind this is not proven in their paper, but they empirically show that the solutions are most often integer. Further, in the rare event that the solutions are non-integer, they "could add an integer constraint and the computational time did not increase noticeably" (Knighton 2005).

Two other situations that are easily considered include employees with varying abilities and shifts requiring different skill sets. The inclusion of both these cases in the formulation is very simple. If an employee is not available to work a shift or does not have the skills to work a shift, the arc connecting that employee to the shift is deleted. This is a very significant feature because it is easy to accomplish and allows for simultaneously scheduling many departments at. According to Knighton, many of the tour-scheduling done in published literature schedules a single department at a time. Scheduling a single department (when some employees could work in more than one)

"could lead a department to over-staff" (Knighton 2005). Further, this change in the system does not result in a loss of integrality of the solution.

The last piece of the formulation is the optimization function. For this, the staff members provided preferences (on a scale of 1 to 100) for each shift. The scheduler provided a "reward" number from 1 to 100 for each shift and employee combination to reflect seniority and job performance. The resulting function is equation (16).

$$\text{MAX} \qquad \sum_{j,k,d,i} (s_{j,k,d}e_i) \cdot \left[pref(s_{j,k,d}e_i) + reward(s_{j,k,d}e_i) \right] \qquad (16)$$

An additional phenomenon required consideration. In the event that the value of the preferences and reward summed to the same amount for two or more available and qualified staff members for a single shift, the network flow model would assign fractional amounts of employees to shifts. In order to correct this infrequent yet potential problem, a separate random number between 0 and 1 was added to each of the $pref(s_{j,k,d}e_i) + reward(s_{j,k,d}e_i)$ to ensure no two values would be exactly the same. Because this random number is between 0 and 1 and the original value is between 2 and 200, its magnitude will not affect the outcome of the problem. Essentially, this step slightly shifts the ties so they are no longer ties.

Application

This solution method was applied to the real world scenario it was meant to model. There were 195 distinct shifts to be covered by the 50 staff members. Because of student privacy, 50 random schedules were generated for the students based on a normal load of three classes per week and each class meeting twice per week. Five separate models were completed; the results are contained in Knighton's dissertation (2005).

The solution time for a given instance was fractions of a second. Because of its efficiency, real-time scheduling for this size of a problem is attainable. The scheduler

can also examine the effect of changes in the schedule or changes in the preferences and manager weightings. Also, by looking at the schedule and seeing which type of staff members are staffed to the full capacity, the scheduler can easily determine which types of employees will most greatly improve the system performance.

After application to the real world problem, the authors show that the formulation is robust enough to handle larger formulations. They generate new problem sets with up to 420 shifts and 100 staff members. The program still solves the problems in "an insignificant amount of computer time" (Knighton 2005).

Conclusion

This formulation by Knighton captures all the elements of a heterogeneous workforce that are necessary in the formulation of the problem of interest in this thesis. It solves the problem in fractions of a second even when the problem instance is very large. This formulation will be the basis for the formulation developed in this thesis.

Airline Based Scheduling Problems

The airlines have received similar focus as other industries regarding personnel scheduling problems. They are faced with similar constraints and work with comparable sizes of staffs. Two papers addressed this problem with differing approaches.

Large Scale Crew Scheduling Problem

Chu, Eric and Ellis address a large scale airline crew scheduling problem (1997). They attempt to schedule crews on a daily basis. They emphasize the method for dealing with the massive number of possible crew pairings. Their main formulation is based on a set partitioning problem. Each column is a pairing of crews and each row is a flight segment. Each segment must be covered by a pairing of crews. The cost of each column is associated with the cost of using that crew for that flight segment.

The solution approach they use consists of three stages. First, linear programs are solved to generate 20 million possible crew pairings (the columns of the overall problem). 2 million of these columns are selected based on the cost of those columns. These 2 million columns are reduced down further to 15,000 using a heuristic to find the 'best' columns. Finally, a branching heuristic is used on the 15,000 column problem to solve the set partitioning problem.

The problem with using heuristics is that the optimal solution is not always found. In the case of this paper, this algorithm performed within 2% of the optimal solution every time it was run. This is a great accomplishment; however, their heuristic did not produce results instantaneously. Unfortunately, they do not publish the average time to completion for their algorithm, but after examining the type of heuristics used and the number of "stages" required, it was likely a time intensive process. Therefore, this research will attempt to maintain total unimodularity therefore removing the necessity to use heuristics but instead use algorithms which ensure efficient solution times.

Airline Cockpit Crew Scheduling Problem

Yan and Chang look at scheduling cockpit crews for a Taiwanese airline (2001). Their problem is defined by having crew members that must be assigned to flights. A feasible schedule must satisfy flight routes, labor union rules, government regulations and the carrier's own policies. A crew is assigned to a series of one or more flights. They must be assigned different sets of duties that meet with the requirements described above. All crew flight paths must start and end at the crew's home base. Additionally, different crew members have different costs and different skill sets.

The authors utilize a network formulation to define the feasible paths a crew could take through the system. Once the network is created, different feasible paths through the network are generated. Each of these paths is called a "pairing" and these

parings are then assigned to different crews to cover all of the tasks that must be completed. A drawback with their formulation is that they necessitate using a branch and bound technique coupled with a heuristic to achieve integer solutions in a reasonable amount of time. This is due to the fact that their side constraints on the network flow problem cause the loss of total unimodularity of the constraint matrix. As previously stated, this research will attempt to maintain total unimodularity of the constraint matrix, therefore not necessitating an exhaustive search or the use of heuristics.

Summary

From the literature, it is clear that a network flow formation of the scheduling problem would be a feasible (and potentially beneficial) method used to solve this problem. The property of total unimodularity should be exploited to its fullest potential as to avoid heuristics and exhaustive searches. In the problem addressed by this thesis, a network flow formation with side constraints will be developed and total unimodularity will be proven to be maintained. This will allow for solutions to be found in a polynomial amount of time with respect to the problem size which will allow AFPC to explore many possible schedules before determining which to use. The methodology utilized to derive this solution is presented in the next chapter.

III. Methodology

Overview

This section will present the solution approach used to solve the initial skills training scheduling problem (ISTSP). First, a discussion about the actual problem and the solution methodology currently in use by AFPC to solve the ISTSP will be presented. This solution methodology will then be compared to a transportation problem to demonstrate that it follows many properties of the transportation problem. Next, a network flow approach to the ISTSP will be presented followed by a discussion total unimodularity and its implications in this formulation. Finally, the algorithm and computer code developed to solve instances of the ISTSP will be discussed.

As was previously discussed, the ISTSP involves assigning students to specific class offerings of various courses. In this formulation, a "course" is defined to be one of the IST courses (as presented in Chapter 1). All officers first graduate from their respective colleges. The graduation dates are spread through the year but are focused mostly in the summer. After graduation, they must be commissioned as officers in the US Air Force. In some cases (such as USAFA graduates), this occurs simultaneously with the graduation from college. For ROTC graduates, the commission is required by law to occur within 365 days of graduation from college. Additionally, graduates from OTS are commissioned immediately upon completing OTS.

After commissioning, all officers must all attend an ASBC class. Each of the classes has a pre-determined start date, graduation date and student capacity. It is also requested that a "blend" of AFSCs and SOCs exist in each ASBC class. This blend is to ensure that students are exposed to officers from other SOCs and AFSCs. Following ASBC, they enter their respective IST sequence of courses which are AFSC dependent. Each IST course has set start dates, graduation dates and maximum capacities. Within

each of these classes, a blend in SOCs should exist. A blend by AFSCs is not necessary because every class is made up of only a single AFSC.

Mathematical Formulation of the Current Methodology

AFPC currently utilizes an algorithm programmed in the SAS/OR Optimization software package. The current algorithm has some short falls that AFPC has asked be addressed in the newly developed algorithm. Since the SAS software package has already been purchased and is in use by AFPC, AFPC has requested that the new algorithm be programmed in this language as well.

The mathematical formulation for the algorithm currently being utilized is:

Sets

P = officers needing scheduling $p \in \mathbf{P}$

A = ASBC classes (including a value for no class) $a \in \mathbf{A}$

F = IFS classes (including a value for no class) $f \in \mathbf{F}$

S = IST (including UPT classes) (including a value for no class) $s \in \mathbf{S}$

Variables

$$x_{p,a,f,s} = \begin{cases} 1 \text{ if officer } p \text{ is assigned to ASBC class } a, \text{ IFS class } f \text{ and IST class } s \\ 0 \text{ otherwise} \end{cases}$$

$v_{p,a,f,s}$ = weight of assigning officer p to ASBC class a, IFS class f and IST class s

Objective Function

$$\text{Maximize} \sum_{p,a,f,u} x_{p,a,f,s} v_{p,a,f,s} \qquad (17)$$

Subject to

$$\sum_{p} x_{p,a,f,s} = 1 \qquad \forall a \in \mathbf{A},\ f \in \mathbf{F}, s \in \mathbf{S} \qquad (18)$$

$$\sum_{p,f,s} x_{p,a,f,s} \le UB_a \qquad \forall a \in \mathbf{A} \qquad (19)$$

$$\sum_{p,a,s} x_{p,a,f,s} \le UB_f \qquad \forall f \in \mathbf{F} \qquad (20)$$

$$\sum_{p,a,f} x_{p,a,f,s} \le UB_s \qquad \forall s \in \mathbf{S} \qquad (21)$$

$$x_{p,a,f,s} \in \{0,1\} \qquad \forall p \in \mathbf{P}, a \in \mathbf{A},\ f \in \mathbf{F}, s \in \mathbf{S} \qquad (22)$$

P is the set of officers which require scheduling. **A** is the set of all available ASBC classes into which the officers from **P** can be scheduled. **F** is the set of IFS classes into which the pilots in **P** can be scheduled. Finally, **S** is the set of all IST classes into which the officers in **P** can be scheduled. In this formulation, UPT is considered to be an individual IST course and the classes of UPT are in the set **S**. Before the optimization is performed on a particular instance, combinations that are not feasible (assigning people to classes outside their career field, assigning people to classes that are in session at the same time, etc.) are eliminated through pre-processing.

The objective function for this formulation is a basic weighted summation. The weights are based solely on the number of courses a particular schedule provides for an individual. For example, if a person has the choice between two schedules where one results in them attending only ASBC and the other results in them attending ASBC and IFS, the second schedule will have a higher weight and will therefore be more desirable.

The subsequent constraints require further explanation. Equation (18) ensures each person is given exactly one assignment of classes. Equations (19), (20) and (21) ensure the population in each individual class does not exceed the number of seats available. Equation (22) simply defines the decision variables as binary variables.

Discussion of the Current Solution Methodology

The problem is essentially a transportation or assignment problem. Each person is assigned a series of classes to attend. Each series is given a weighted value and the overall value is maximized. However, there are many more choices for assignment than there are people to be assigned to them. By the definition of an assignment problem the number of people or "machines" must equal the number of tasks to be performed. Since this is not true in the current methodology, this problem is described by the more general transportation problem.

Because this problem is a transportation problem with a totally unimodular constraint matrix, it is in the class **P** and can be solved in a polynomial amount of time (Bazaraa, Jarvis and Sherali 2005). In the current formulation, the weights (v and w) are chosen solely on the number of classes that are assigned to an officer. The objective function does not consider the time between those assigned classes. Allowing officers to skip courses has been determined to be an unacceptable practice; however, the algorithm currently in use allows this policy but applies a penalty to the objective function in these cases. A final problem with this algorithm is that there is no mention or consideration for blending the SOC and AFSC of students in classes. This has become a top priority for the decision makers, but is not evident in the application of the solution methodology. All of these limitations are addressed in the newly developed solution algorithm for the ISTSP.

Mathematical Formulation of the Network Flow ISTSP Algorithm

The network flow initial skills training scheduling problem (NFISTSP) algorithm is formulated as follows:

Sets

$\mathbf{S}$ = set of all commissioning sources = $\{s_1, s_2, ..., s_n\}$

$\mathbf{A}$ = set of all AFSCs = $\{a_1, a_2, ..., a_m\}$

$\mathbf{T}$ = set of all training courses = $\{t_1, t_2, ..., t_k\}$

$\mathbf{C}_t$ = set of all classes of training course t = $\{c_{t_1}, c_{t_2}, ..., c_{t_l}\}$

Variables

x_{s,a,c_t,c'_t} = number of officers from commissioning source s in AFSC a going from class c_t to class c'_t

Parameters

d_{c_t,c'_t} = number of days bettween class c_t and class c'_t

y_{s,a,c_t} = number of officers from commissioning source s in AFSC a graduating from class c_t

Objective Function

$$\text{Minimize} \qquad Z = \sum_{\forall t \in T} \sum_{\forall c'_t \in \mathbf{C}_{t+1}} \sum_{\forall c_t \in \mathbf{C}_t} d_{c_t,c'_t} \sum_{\forall a \in \mathbf{A}} \sum_{\forall s \in \mathbf{S}} x_{s,a,c_t,c'_t} \tag{23}$$

Subject to

$$y_{s,a,c_1} = \sum_{\forall c' \in \mathbf{C}_2} x_{s,a,c_1,c'_2} \qquad \forall s \in \mathbf{S}, a \in \mathbf{A}, c_1 \in \mathbf{C}_1 \tag{24}$$

$$\sum_{\forall c_t \in \mathbf{C}_t} x_{s,a,c_t,c'_{t+1}} = \sum_{\forall c'_{t+2} \in \mathbf{C}_{t+2}} x_{s,a,c_{t+1},c'_{+2}} \; \forall s \in \mathbf{S}, a \in \mathbf{A}, c'_{t+1} = c_{t+1} \in \mathbf{C}_{t+1}, t = 1, 2, ..., n-1 \tag{25}$$

$$\sum_{\forall s \in \mathbf{S}} \sum_{\forall a \in \mathbf{A}} \sum_{\forall c_k \in \mathbf{C}_k} x_{s,a,c_k,c'} = \sum_{\forall s \in \mathbf{S}} \sum_{\forall a \in \mathbf{A}} \sum_{\forall c_t \in \mathbf{C}_1} y_{s,a,c_1} \qquad c' = \text{"sink"} \tag{26}$$

$$\sum_{\forall s \in \mathbf{S}} \sum_{a \in \mathbf{A}'} \sum_{\forall c_t \in \mathbf{C}_t} x_{s,a,c_t,c'_{t+1}} \le UB_{c'_{t+1}} \qquad \forall c'_{t+1} \in \mathbf{C}_{t+1}, t = 1, 2, ..., n-1 \tag{27}$$

$\mathbf{A}'$ is defined as the set of AFSCs that are allowed in c'_{t+1}

$$\sum_{\forall s \in \mathbf{S}} \sum_{\forall c_t \in \mathbf{C}_t} x_{s,a,c_t,c'} \le UB_{a,c'_2} \qquad \forall a \in \mathbf{A}, c'_2 \in \mathbf{C}_2 \qquad (28)$$

$$\sum_{\forall a \in \mathbf{A}'} \sum_{\forall c_t \in \mathbf{C}_t} x_{s,a,c_t,c'_{t+1}} \le UB_{s,c'_{t+1}} \qquad \forall s \in \mathbf{S}, c' \in \mathbf{C}_{t+1}, t = 1, 2, ..., n-1 \qquad (29)$$

Another helpful aspect of network flow problems is that they can be seen in graphical form. Figure 2 presents the general network flow model in pictorial form.

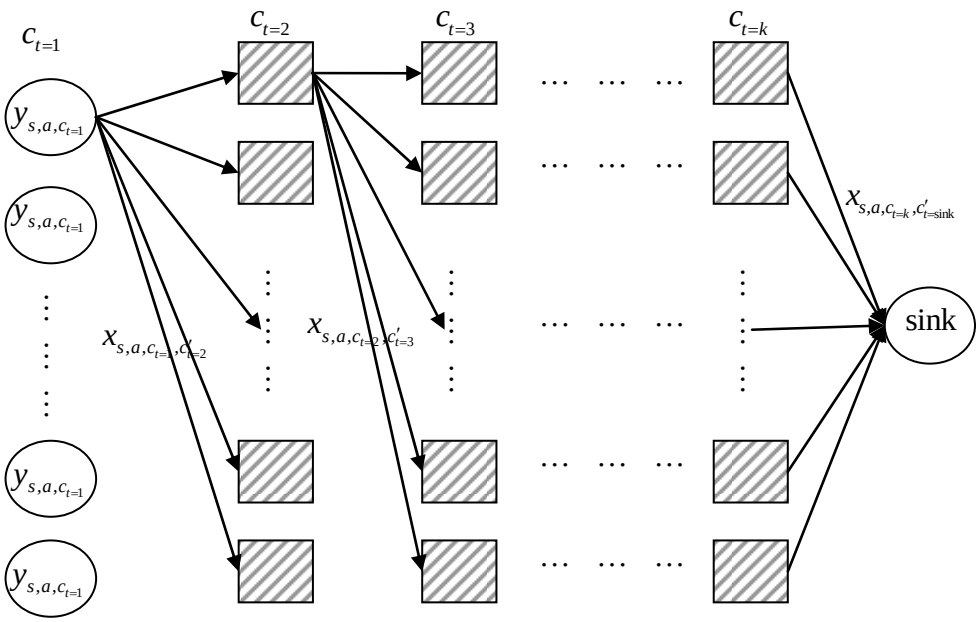

Figure 2: General Network Flow Formulation

The circles on the left represent the source nodes from which all newly commissioned officers originate. This is the first "course" in the set **T**. The first column of squares represents the next course in the sequence. Each square is a specific *class* in that course. They flow from course to course in this manner. Additional arcs connect the classes; only a few are displayed to enhance readability. The circle on the right in Figure 2 is the sink node to which the last set of classes must flow.

Although Figure 2 presents a network flow representation for the ISTSP, it is not complete. This is because each of the classes is not simply a node. It is really a collection of nodes. Figure 3 illustrates the information which is contained in each collection of nodes.

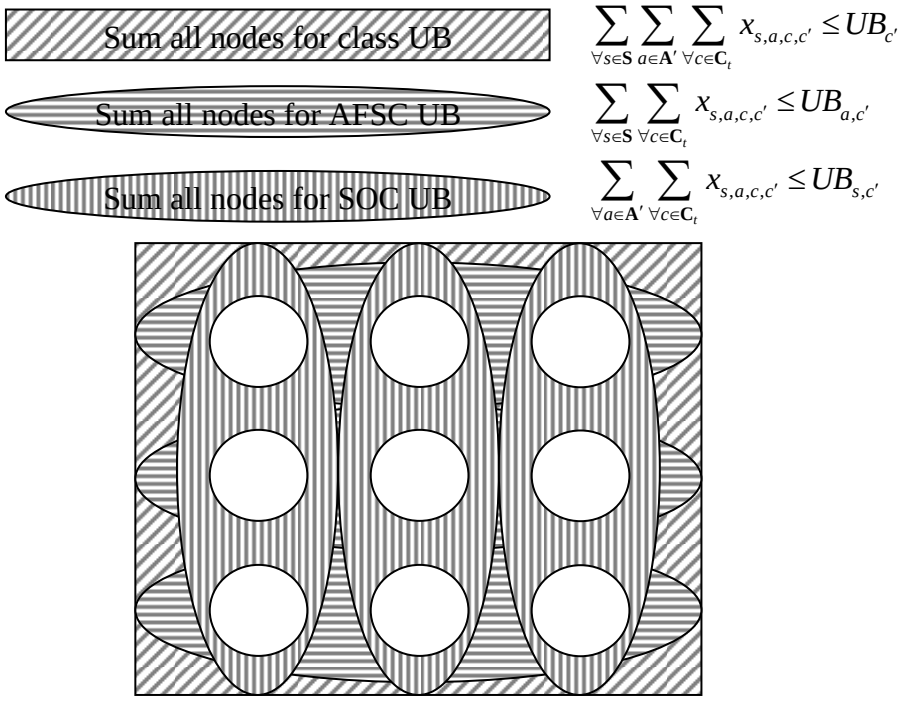

Figure 3: Class Node Breakdown

Each of the circles actually represents a node in the directed graph. The square represents a combination of nodes; it corresponds to the first constraint listed in Figure 3. The flows on arcs which terminate at any of those classes are combined; this summation must be less than the capacity at the class. The horizontal ovals encompass the nodes that have the same SOC. The flows on arcs which terminate at any of the nodes in this set (same SOC and class) are combined; this summation must be less than some upper bound which is a percentage of the total class capacity. Finally, the vertical ovals indicate that these nodes share the same AFSC. As with the previous two sets, the flows on arcs which terminate at any of the nodes in this set (same AFSC and class) are combined, and this summation must be less than some upper bound. An arc in Figure 2 is really originating and terminating at nodes with the same SOC and AFSC that are combined into sets of different class offerings of different courses. In the mathematical

formulation, the number of SOCs and AFSCs are unconstrained (i.e they are not limited to 3 of each as shown in Figure 3). In the ISTSP instances solved for this research (results of which are presented in Chapter 4), there were 3 SOCs and 6 AFSCs.

The network formulation may look complicated, but should be easily understood with a small amount of explanation. There are essentially a number of layered networks (one for each combination of SOC and AFSC) whose nodes can be grouped into sets with similar qualities. These separate networks control the officers in the corresponding AFSC and from the corresponding commissioning source. Each person is assigned to a class in every single course. It is the $d_{c,c'}$ parameters (number of days between graduation of class c and start of class c') that drive the optimization. It is necessary to note that a student could be assigned to a course that they are not required to (and actually cannot) take (for example, a pilot to PRIM). In these cases, the number of down days between the previous course and this course is given a value of 0. Thus, the objective function value for NFISTSP is not affected by the assignment of students to these courses. Additionally, since the student's AFSC and the AFSC required for the course does not match, the students are not counted in the number of students enrolled in the course.

When a student is assigned to a class required for their IST sequence, the d parameter corresponds to the number of ineffective days of training between the courses. Therefore, the program will seek to minimize those ineffective days. The network is defined by the basic flow balance equations that control any network. Equation 24 is the flow balance for the first set of nodes (the supply nodes). Equation 25 is the flow balance for the transshipment nodes between all the classes. Equation 26 is the flow balance for the last node (the sink node). Every student will terminate the network at the same sink node.

The separate networks are linked with the introduction of capacity constraints (equations 27, 28 and 29). Equation 27 ensures that each class is kept under the class's total capacity. Equation 28 ensures that the number of officers from a particular AFSC assigned to a particular class is under a limit (in this case, a percentage of a class's total capacity). Equation 29 ensures that the number of officers from a particular SOC in a particular class is under a limit (again, a percentage of a class's total capacity). These upper bound constraints ensure that there is a mix of AFSCs and commissioning sources in ASBC and a mix of only commissioning sources in all other classes.

For specific instances of the ISTSP solved in this research, AETC has stated that as long as only a maximum of a certain percentage of the population of each class share the same SOC or AFSC then the class will be sufficiently mixed. Currently, that number is set at 75%, however, the customer will regularly change this (either in actual application or to simply explore the effects on the schedule if it were changed).

Discussion of the NFISTSP Formulation

The NFISTSP formulation is very similar to a transportation problem with transshipment nodes (Bazaraa, Jarvis and Sherali 2005). The only difference is the constraints regarding the blending of classes. In the traditional transportation problem, there is no mention of capacities being placed on a collection of arcs which describes the restrictions enforced by the blending constraints. With these additional constraints, NFISTSP risks the loss of constraint matrix total unimodularity (which, as previously described, is essential to timely execution of the algorithm). Fortunately, as will be explained in the next section, this property is not lost.

Another useful aspect of the NFISTSP formulation is its generality. In this case, the people being scheduled were heterogeneous; they have different SOCs and different AFSCs. The NFISTSP formulation really allows for any number of these two attributes.

This generality could prove useful in other personnel scheduling problems where the heterogeneousness of the population is more complex.

Total Unimodularity

As discussed earlier, total unimodularity is a property which ensures integer programs can be solved in polynomial time. Total unimodularity is the reason the transportation (with and without transshipment nodes), assignment and minimum cost network flow problems are in class *P* (Bazaraa, Jarvis and Sherali 2005). In the NFISTSP previously described, the additional constraints are used to account for the blending of AFSCs and SOCs.

The NFISTSP formulation is totally unimodular and hence the problem can be solved in a polynomial amount of time. The reason for this relies on the network flow problem structure and a theorem proven by Nemhauser and Woosley (1999). They prove that the following two statements are equivalent:

1. *A* is totally unimodular.
2. For every $J \subseteq N=\{1,\ldots,n\}$, there exists a partition J_1, J_2 of J such that

$$\left| \sum_{j \in J_1} a_{ij} - \sum_{j \in J_2} a_{ij} \right| \leq 1 \qquad \forall i = 1, \ldots, m$$

It is also useful to note that if matrix Q is totally unimodular then Q^T is totally unimodular. Therefore, without loss of generality, it is valid to assume that the indices *i* or *j* in the previous theorem could refer to columns or rows. Another important property involves deleting rows or columns out of a totally unimodular matrix. By definition, a totally unimodular matrix requires the determinants of all square sub matricies be 0, 1 or -1. If a matrix Q is totally unimodular, then deleting a row or column out of Q would only delete the sub matrices that included that row or column. All other square sub matrices would remain unchanged. Since the remaining sub matrices are left unchanged, their

determinants would still be 0, 1 or -1 and the remaining matrix would still be totally unimodular.

For example, Figure 4 is a generic matrix which is assumed to be totally unimodular. Since it is totally unimodular, all square sub matrices must have a determinant of 0, 1 or -1. Since each individual entry is a square sub matrix, it is apparent that each a_{ij} has a value of 0, 1, or -1. If the first row is removed, the only square sub matrices that are affected are the ones that include elements from the first row. All other square sub matrices remain unchanged and therefore still have a determinant of 0, 1 or -1. To examine the sub matrices that are changed, let us look at an example.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1j} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2j} \\ a_{31} & a_{32} & a_{33} & a_{34} & \cdots & a_{3j} \\ a_{41} & a_{42} & a_{43} & a_{44} & \cdots & a_{4j} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{i1} & a_{i2} & a_{i3} & a_{i4} & \cdots & a_{ij} \end{bmatrix}$$

Figure 4: Example Matrix

The square sub matrix formed by elements a_{11}, a_{12}, a_{21}, and a_{22} , which is given by $\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$, would be affected by removing row 1; in fact, it would no longer exist. Instead, two square sub matrices are created: a_{21} and a_{22}. However, these square sub matrices already existed in the original matrix and therefore have a determinant of 0, 1 or

-1. Similar logic can be used for all square sub matrices to see that the property of total unimodularity is not lost when the first row is deleted.

Consider the NFISTSP formulation previously mentioned. A will be defined as the node arc incidence matrix of the NFISTSP; this matrix simply defines the source and destination nodes for each arc. Since the columns refer to different arcs, and each arc has only one origin node and one destination node, each column will contain exactly one value that is 1 and one value that is -1; all other values are 0. Therefore, it is apparent that each of the columns of A sum to 0. The rows of A are assigned into the set J_1 described in the theorem above.

Next, consider the matrix of side constraints C. All side constraints in the problem are summations of arcs set less than some upper bound. The vector of coefficients for a constraint of this type is simply a vector of 1s and 0s (a 1 if the arc is included in the sum, a 0 if not). Therefore, the entire matrix of C is comprised only of 0s and 1s. We will also define C' as the compliment of C (that is, if a value in the C matrix is 1, the value of the same position in C' is 0 and vice-versa). C' is not necessary in the formulation of the problem, but is simply a tool used for the proof that the formulation is totally unimodular. C_i is defined as row i of C. It is clear that the following statement is true:

$$C_i + C_i' = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix} \qquad \forall i$$

Therefore, assign all odd rows of C and C' into J_1 and all even rows into J_2. Only two potential cases could exist: 1) there is an even number of rows in C and C' or there an odd number of rows in C and C'. In either case, the rows from the matrix A that are in J_1 will sum to 0. The remaining rows in J_1 are the odd rows of C and C'. In the case where there is an even number of rows, these rows will sum to $\frac{m}{2}$ (where m is the

number of rows in C) because $C_i + C_i' = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix} \forall i$. Summing up the rows in J_1 will yield the same result. Therefore $\left| \sum_{j \in J_1} a_{ij} - \sum_{j \in J_2} a_{ij} \right| = 0$ for every column.

In the event that there is an odd number of rows, the $\sum_{j \in J_1} a_{ij}$ would total to 1 more than $\sum_{j \in J_2} a_{ij}$ resulting in $\left| \sum_{j \in J_1} a_{ij} - \sum_{j \in J_2} a_{ij} \right| = 1$. In either case, the value rule that $\left| \sum_{j \in J_1} a_{ij} - \sum_{j \in J_2} a_{ij} \right| \leq 1 \; \forall i = 1, ..., m$ is satisfied. Therefore, the matrix $\begin{bmatrix} A \\ C \\ C' \end{bmatrix}$ is proven to be totally unimodular.

Further, C' can be removed without destroying the total unimodularity of the matrix. As stated earlier, the removal of a row in a totally unimodular matrix does not remove the property of total unimodularity. Therefore, after removing the rows in C', $\begin{bmatrix} A \\ C \end{bmatrix}$ is still totally unimodular.

The final consideration is the arc capacities. For each row associated with an arc capacity, there will only exist a single 1; the remaining values will be 0. Additionally, there will only exist a single 1 in each column. The constraints can therefore be rearranged to form an identity matrix. When these constraints are added to the problem, the matrix becomes $\begin{bmatrix} A \\ C \\ I \end{bmatrix}$ where I is the identity matrix. Nemhauser and Woosley state

that if a matrix Q is totally unimodular, then $\begin{bmatrix} Q \\ I \end{bmatrix}$ is totally unimodular. Replacing Q with $\begin{bmatrix} A \\ C \end{bmatrix}$, the proof that NFISTSP is totally unimodular is complete.

Software Programming

The NFISTSP algorithm was written in SAS using the SAS/ACCESS and SAS/OR packages along with the basic SAS commands. This program was chosen because the problem size was too large for simpler solver programs (such as Microsoft Excel Solver). Even with Frontline's Premium Solver for Microsoft Excel, the variable and constraint limits would have been exceeded in the instances used to demonstrate the algorithmic efficiency and effectiveness. Therefore, a more robust and powerful software platform was needed. Since the end-user (AFPC) had already purchased SAS and the necessary additional packages (SAS/ACCESS and SAS/OR), the NFISTSP algorithm could be incorporated without incurring an additional cost.

Since the current methodology was written in SAS, the customer is very familiar and comfortable with using it. The interface is based on Microsoft Excel tables that are generated by the user. With this easy to use interface, the execution of the NFISTSP algorithm can be accomplished by personnel not familiar with the code utilized to create it. However, changes to the code (not already built into the robustness of the program) will require personnel who have experience using SAS. For example, a method is already designed in NFISTSP for users to define class blends. In this case, it is simply a number in one of the Microsoft Excel sheets used as inputs. Changing these values could be accomplished by any end-user of the algorithm. However, if a structure change, such as allowing people to skip classes, requires implementation, personnel with SAS experience will be required to alter the algorithm.

The following is the pseudo-code generated from the NFISTSP algorithm. This implementation differs slightly from the NFISTSP formulation previously discussed. In the formulation, every student was assigned to every course (regardless of the necessity to take the course). The NFISTSP formulation is a general case; the algorithm discussed below takes advantage of the problem structure and eliminates the unnecessary and infeasible arcs. There were a small number of commissioning sources (three) and AFSCs (six). Therefore, a more specific code was created to demonstrate efficiency. The actual SAS code is presented Appendix A.

Pseudo-Code of the Program

- Read in the Microsoft Excel sheets as inputs
    - People lists
    - ASBC class lists
    - IFS class lists
    - IFT class lists
- Format the people list
    - Change the AFSC column into appropriate symbols
    - Delete unnecessary entries
- Create the source nodes
    - Group people together into “pools” rather than individuals
- Create trans-shipment nodes
    - Create ASBC nodes (including a “skip” node)
    - Create IFT nodes (including a “skip” node)
    - Create IFS nodes (including a “skip” node)
    - Create UPT nodes (including a “skip” node)

- Create arcs connecting nodes
    - Create Arcs from source nodes to ASBC nodes
    - Create Arcs from the ASBC nodes to the IFT nodes
    - Create Arcs from the ASBC nodes to the IFS nodes
    - Create Arcs from the IFT nodes to the UPT nodes
    - Create Arcs from the IFS nodes to the “sink”
    - Create Arcs from the UPT nodes to the “sink”
    - Create Arcs from the “skip” nodes at each set of classes to the “sink”
- Create side constraints for AFSC, SOC and class capacities for each class
    - Create side constraints for ASBC
    - Create side constraints for IFS
    - Create side constraints for IFT
    - Create side constraints for UPT
- Square the costs on each arc in order to encourage an even spread of down days.
- Read the nodes, arcs and constraints into the “Proc Netflow” function
- Create Output tables from the data output by the “Proc Netflow” function

Summary

It is clear that the solution currently in place does not meet the requirements set forth by Air Force senior leadership. In response to their new demands, the NFISTSP formulation was developed. This formulation is robust enough to handle the changing nature of the Air Force and is mathematically strong enough to maintain total unimodularity in the constraint matrix in order to ensure the optimal solution can be reached in a polynomial amount of time. The algorithm is written in SAS/OR to ease the implementation of NFISTSP by AFPC.

IV. Results

Overview

In this section, the results produced by the developed algorithm will be discussed. First, the inputs used to create the schedules will be explained. After that, the outputs from the produced schedules will be described. Next, examples of outputs will be shown. Finally, the schedules that were actually produced will be compared in order to determine how different inputs affect the schedules and the discussed outputs from those schedules.

Required Inputs

A single input file details the students to be scheduled as well as the courses which they require. In the current method, students are scheduled to courses at multiple times throughout the calendar year. As a result, some people were scheduled for a portion or all of their course sequence while others were not scheduled at all because there was insufficient capacity for classes during the times they were available. In the network flow approach, it is assumed that all courses are scheduled simultaneously. This required removing any partially or non-scheduled students from the schedule derived using the previous technique and generating a new (and complete) schedule. Additional input files list the class offerings (including capacity and start and completion dates) for each course.

Generating Data Sets

The input files utilized to demonstrate NFISTSP's effectiveness and efficiency were generated based upon real data sets supplied by AFPC. Unfortunately, several courses do not generate an annual (or even semi-annual) class offering. Instead, a notice for a class offering may be given to AFPC for scheduling immediately before the class is to begin. Obviously, this makes scheduling efforts extremely difficult. This "short notice" scheduling will not be acceptable with NFISTSP. Instead, course directors will

be required to generate a listing of their classes well in advance of the actual class start date.

Some extrapolations were required to ensure completeness of the data sets. For instance, a list of ASBC classes which were available in 2007 is available and the list of class offerings for PRIM ranged from 2006 to 2009. It will not be possible to schedule students into courses which predate the ASBC classes. To circumvent this problem, a class listing was generated for a *given* year. If actual information for a course's class offering was not available for the selected year, extrapolations were made from any relevant class information available. The start/completion dates may not directly coincide with the actual start/completion dates, but the *interval* between classes within a course corresponds to the data sets which were made available for this research.

Since the data sets are realistically generated from past years' information, the scheduling results may not exactly coincide with any upcoming years. However, the schedules are indicative of how different choices (in the form of problem parameters) will affect an overall outcome. The choices which the decision maker must make regarding the parameter settings are numerous and are often changing. Therefore, a fast, repeatable method for generating schedules is desired. After the parameter settings have been determined, the final product will be delivered to AFPC and AETC for implementation on the actual student and class data sets.

Parameter Inputs

AFPC would like to determine the effect certain Air Force policies have on the NFISTSP. These policies include the amount of post graduation leave granted to USAFA graduates, the amount of commissioning delay for ROTC graduates and forcing class blends of SOC and AFSC. By law, USAFA graduates are permitted 60 days of leave to be used immediately after graduation. They can voluntarily waive their rights to this

leave, but it is considered to be “use or lose” leave. This implies that if this leave is waived, the USAFA graduates do not get to keep it for later use. The possible change is to ask USAFA graduates to waive their leave in order to attend an early class offering of ASBC, thereby enabling them to begin their IST course sequence earlier. For graduates that want to get into their career fields as soon as possible, this option may be beneficial.

After graduation from college for an ROTC cadet, congressionally mandated law dictates that they are to be commissioned in 365 days or less. Current Air Force policy states that it will be 180 days or less. Until recently (within the last few years), the practice had been to commission the ROTC graduates immediately upon their graduation from their respective colleges. By allowing the NFISTSP algorithm to choose the commissioning date, the algorithm is given more opportunity to improve the objective function.

Finally, the class blend levels are examined to determine how necessary they are and what effects they have on the schedule. The blends are defined as upper bounds on the number of seats in a class that can be assigned to a certain AFSC or SOC. The available parameter settings should be examined at multiple levels to determine the preferred settings. In addition, all combinations of these settings should be explored. The formulation allows for numerous parameters to be modified to any level, but the parameters mentioned are of most interest. Each of these parameters was explored at 3 levels, which creates 27 distinct combinations of parameter settings. The actual values used for each of the parameters are:

1. Minimum days of post USAFA leave (L): 60, 30 and 0
2. Maximum days of ROTC commissioning delay (C): 0, 180 and 365
3. Maximum % of a class allowed to be a certain SOC or AFSE (B): 50%, 75% and 100% (100% corresponds to No Forced Blend)

The letters in parentheses are used as labels in the following discussion. These settings are listed in order of most restrictive to least restrictive for a given parameter. Therefore, the least restrictive combination is (L) at 0, (C) at 365 and (B) at 100% while the most restrictive is (L) at 60, (C) at 0 and (B) at 50%. Beyond those three parameters, two objective function calculations were considered. The first method involved minimizing the total number of down days for all students (MIN TDD). The problem with this method is that there is no consideration for balancing the schedule between people. For example, this method would not distinguish between the following two examples:

1. Send one person on an arc to wait 0 days and send another person on an arc to wait 90 days
2. Send both of them on an arc to wait 45 days

Given this choice, the senior Air Force leadership would prefer to have NFISTSP choose option 2. This is because option 2 attempts to equalize the average number of down days rather than just minimizing the total number of down days.

To account for this, a second objective function calculation involves utilizing the difference in days squared as an objective function. This method more equally distributes the number of down days across all students, and is therefore labeled DIST DD. In this method, NFISTSP still minimizes wasted days, but it would also equalize the average number of down days rather than simply minimizing the total number of down days. In the above example, it would choose the second choice.

The problem with the DIST DD method is that certain situations may produce sub-optimal solutions. For example, suppose the following alternatives were encountered:

1. Send one person to wait 3 days and another person to wait 4 days

2. Send one person to wait 1 day and another person to wait 5 days

In this case, option 1 creates an average wait of 3.5 days while option 2 creates an average wait of 3 days. Option 2 is the optimal choice. However, if the values are squared (to more evenly distribute the down days), the options become:

1. Send one person to wait 9 $days^2$ and another person to wait 16 $days^2$
2. Send one person to wait 1 $days^2$ and another person to wait 25 $days^2$

In this case, option 1 creates an average wait of 12.5 $days^2$ while option 2 creates an average wait of 13 $days^2$. According to the squared method, option 1 is the better choice. Both of these situations are encountered several times in a schedule, therefore either method could create a "better" schedule (depending upon the definition of "better"). For this reason, both alternatives will be explored.

A potential variant of solving two different objective functions could be to run the MIN TDD method first, then incorporate this solution as a constraint in the DIST DD method. This would ensure optimality would be maintained *and* the total number of ineffective days spent would be more evenly distributed among all students. Unfortunately, implementation of this method is not efficient because the coefficients of the additional constraint are not guaranteed to be either 0, 1 or -1 which would potentially destroy the total unimodularity of the constraint matrix.

Because two objective function calculation methods are evaluated, the total number of possible scenarios to consider is 54. In each of the 54 scenarios, a series of outputs are needed to evaluate the schedule produced.

Outputs

The outputs generated by NFISTSP are flows (in terms of students sent from one node to another) across each arc. The total time for the algorithm to run is between 3 and 4 minutes. However, most of that time is pre-processing. The actual network simplex

algorithm takes an average of 22 seconds to complete. When the NFISTSP algorithm completes, a list of arcs is presented with values detailing how many students should be sent on an arc (or to a class offering of a course). Each arc has costs (in terms of number of down days) associated with them. The output table listing the individual arcs can be readily viewed in a spread sheet (such as Microsoft Excel). The output data is accessible via pivot tables which enable users to view desirable course and class statistics.

To demonstrate the effectiveness and efficiency of NFISTSP, a collection of results under varying scenarios was generated. These results enable the decision maker to view the generated schedules and decide which one is preferable (i.e. which parameter settings are best). Unfortunately, it is not feasible for a decision maker to understand every individual student's schedule and make a decision which considers every student. Instead, a few metrics exist that, when considered as a whole, adequately summarize the overall performance of the schedule. These metrics include:

- Average days ineffective training for each AFSC and SOC combination
- Maximum number of ineffective training days for an individual
- Number of people not scheduled for required training
- Average amount of leave given to USAFA graduates before training begins
- Minimum amount of leave given to USAFA graduates before training begins
- Average number of days an ROTC graduate waits for commissioning after graduation
- Maximum number of days an ROTC graduate waits for commissioning after graduation

These metrics attempt to capture all aspects which the Air Force's senior leadership is trying to balance. Each metric is collected in a different manner: some are

computed by NFISTSP as it generates a schedule, and some are computed after the output has been viewed in Microsoft Excel. For the *average days of ineffective training* for each combination of AFSC and SOC, simple mathematical operations and pivot tables in Microsoft Excel were utilized. For each feasible arc (i.e. an arc whose terminal node has a class start date after the origin node's class completion date), the total number of down days was computed by determining the product of the cost for each arc (i.e. number of down days) and the number of students assigned to the arc. The arcs with AFSC, SOC and class offering of a course in common were combined to give the total number of students in a class. The individual arcs were then divided by this total to determine the average number of down days for a given AFSC and SOC in a class offering of a course. Finally, the averages for each course an AFSC must take were combined to get the total average number of down days. When the number of students required to attend a course exceeds the capacity of all classes, some students are not able to attend the course. In this case, the number of down days was only included for courses which the students actually attended.

Calculating the *highest number of down days given assigned to an individual in the optimal solution* was actually quite difficult because the algorithm does not assign *individuals* to classes but rather assigns *groups* of students to classes. The advantage of utilizing a network flow formulation for NFISTSP is that all potential trade offs between different alternatives are considered. As a result, the "worst" path may not lie in the summation of all of the worst possible arcs (because these options are avoided if possible), but instead in the summation of a few "poor" arcs. Therefore, a similar network flow approach must be used to find the actual worst path taken. To accomplish this task, the NFISTSP output was examined. Any arc which required skipping a course or had zero students utilizing it was removed from consideration; the remaining arcs had

greater than zero flow (in terms of students). Therefore, the network remaining consists only of utilized arcs that resulted in a person being fully scheduled. From here, another "source" node was added. The original source nodes become transshipment nodes. The new source node has arcs which terminate in the nodes which represent the commissioning sources. This will allow for a supply of 1 to be put on this new source node and then network can choose to which of the commissioning source nodes this single supply will go. When the single supply is added, the network flow function that is built into SAS is run again with some small variations. Instead of searching for the MINIMUM cost network flow, it is run searching for the MAXIMUM cost network flow. With this procedure, resultant path is the worst possible path that was actually taken in the optimal solution to the NFISTSP.

A count of the *number of students not attending all required courses* was generated by NFISTSP and combined by AFSC and SOC using Microsoft Excel. The last set of measures relates to the *number of days between graduation and the start of ASBC* for USAFA graduates and the *number of days between graduation and commissioning* for ROTC graduates. To determine the average number of days on leave or delayed in commissioning, a similar approach was taken as described in finding the average number of ineffective days. First, the number of students who had delays or leave (depending on SOC) were calculated, then the number of days delayed or on leave for each student was determined (the maximum and minimum of these values per AFSC and SOC, respectively, were taken as measurements). These values were then multiplied together and summed by AFSC and SOC (this resulted in finding the total number of "leave days" or "delay days" per AFSC and SOC). Next, the number of people in the same AFSC and SOC were summed up to find the number of people in a given AFSC or SOC. The last step involved dividing the total number of days delayed or on leave by the

number of people that shared those days. This resulted in the average number of days on leave or delayed commissioning.

A pivot table and graph can be generated for viewing individual classes. This was not specifically requested by AFPC, but it can be utilized to determine the composition of a specific schedule.

Example Output

The following are examples of the data that can be found in a single replication of a scenario. The first graph shows the composition of all ASBC courses by SOC. This is generated from a pivot table and graph. It would not be feasible for a decision maker to view a similar graph for every possible schedule, but this tool displays, in an easily readable form, any scenarios of interest.

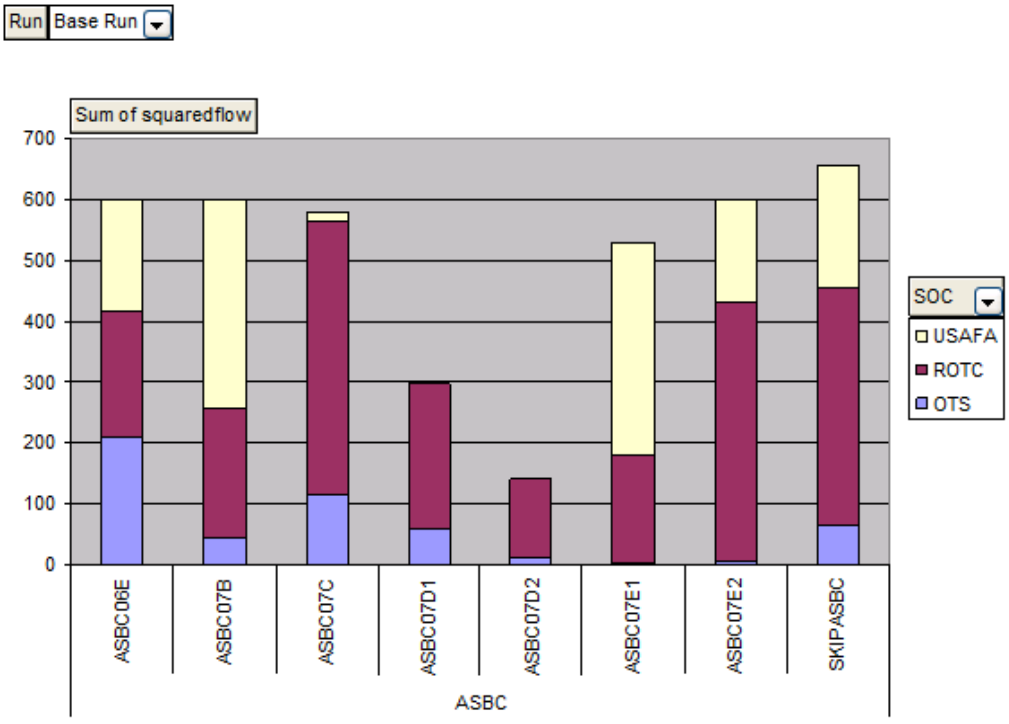

Figure 5: Example Class View comparing SOC Blend

It is apparent that there are empty seats in ASBC07D1 and ASBC07D2 and that other students are being assigned to skip the ASBC course. This is due to the fact that the ASBC07D2 begins prior to students (particularly USAFA graduates) becoming available. The students either have not yet graduated or been commissioned, or they are not available due to USAFA leave rules.

This next figure shows the same ASBC classes but includes a breakdown by the AFSCs that comprise them rather than the SOC.

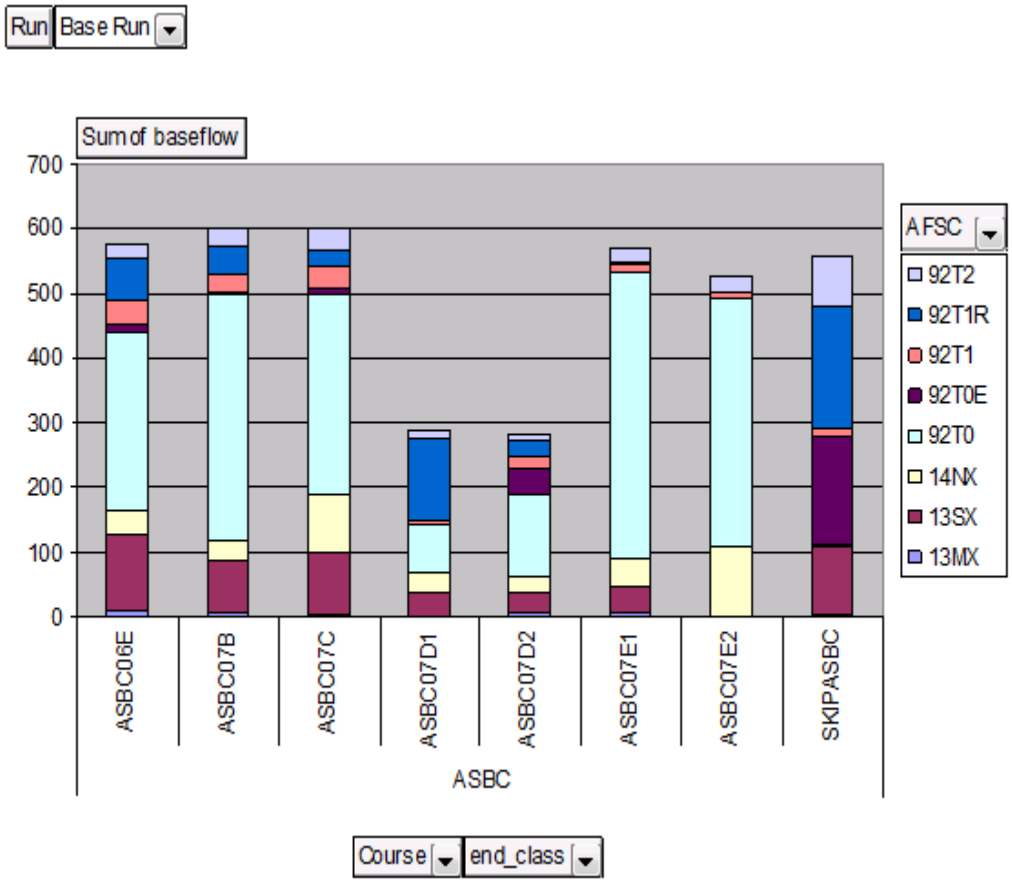

Figure 6: Example Class View comparing AFSC Blend

As shown in Figures 5 and 6, an upper bound class composition of 75% of a given AFSC and SOC can create a schedule with an adequate class blend. Graphics such as these are very useful in depicting the outcomes of choices to the decision maker.

Unfortunately, it is not practical to present graphs for all 54 scenarios. Instead, the outputs previously described will be presented in tabular form. The output for a *single* scenario schedule is presented in Table 2.

		OTS		ROTC		USAFA	
	Objective Function	MIN TDD	DIST DD	MIN TDD	DIST DD	MIN TDD	DIST DD
13MX	Average Days Through	147.7	159.3	163.7	159.9	50.2	66.6
	Number Of People Skip	0	0	2	2	2	2
	Average Initial Days	N/A	N/A	0	0	60	60
	Extreme Initial Days	N/A	N/A	0	0	60	60
13SX	Average Days Through	161.7	150.8	171.2	173.8	113.0	115.1
	Number Of People Skip	0	0	76	76	30	30
	Average Initial Days	N/A	N/A	0	0	60	60
	Extreme Initial Days	N/A	N/A	0	0	60	60
14NX	Average Days Through	672.8	299.5	366.0	360.7	215.6	310.6
	Number Of People Skip	0	0	0	0	0	0
	Average Initial Days	N/A	N/A	0	0	60	60
	Extreme Initial Days	N/A	N/A	0	0	60	60
92T0	Average Days Through	367.3	285.7	285.5	271.3	236.0	267.2
	Number Of People Skip	0	0	2	2	0	0
	Average Initial Days	N/A	N/A	0	0	60	60
	Extreme Initial Days	N/A	N/A	0	0	60	60
92T0E	Average Days Through	310	335	0	0	0	0
	Number Of People Skip	0	0	115	115	103	103
	Average Initial Days	N/A	N/A	0	0	60	60
	Extreme Initial Days	N/A	N/A	0	0	60	60
92T1	Average Days Through	325.4	301.8	208.7	221.4	0	0
	Number Of People Skip	0	0	0	0	15	15
	Average Initial Days	N/A	N/A	0	0	60	60
	Extreme Initial Days	N/A	N/A	0	0	60	60
92T1R	Average Days Through	236.9	250.0	0	0	0	0
	Number Of People Skip	1	1	329	329	26	26
	Average Initial Days	N/A	N/A	0	0	60	60
	Extreme Initial Days	N/A	N/A	0	0	60	60
92T2	Average Days Through	104.9	108.4	129.2	119.2	82.7	101
	Number Of People Skip	45	45	34	34	0	0
	Average Initial Days	N/A	N/A	0	0	60	60
	Extreme Initial Days	N/A	N/A	0	0	60	60

	AFSC	SOC	Days
Base Max Solution	14NX	ROTC	1057
Square Max Solution	14NX	ROTC	526

Table 2: Sample Output

Each measure is broken down by AFSC, SOC and the type of objective function utilized. The objective functions are based upon the weights of the network flow arcs. In the objective function which minimizes total down days (MIN TDD), the arc weights are simply the number of ineffective days between class offerings of different courses. In the objective function which distributes the down days (DIST DD), the arc weights are the square of the number of ineffective days between class offerings of different courses. The first measure, *Average Days Through*, measures the average number of days a person spends waiting in order to complete their training. If they cannot complete training due to a lack of available seats, the number of days spent waiting for the training they did complete is counted. The next measure (*Number of People who Skip)* shows the number of people that could not complete training due to a lack of available seats. The third measure (*Average Initial Days*) is the average number of days until the officer is commissioned (for ROTC graduates) or the average number of days of post graduation leave (for USAFA graduates). The final measure is either the *maximum number of days* a person would wait for a commission (for ROTC graduates) or the *minimum amount of leave* they would receive (for the USAFA graduates).

These graphs and tables increase understanding of individual classes and allow decision makers to gain an in depth look at a particular schedule. Again, it is not practical to suggest that a decision maker view all of this information to make a decision. Therefore, the output is abbreviated further. The final output table summarizing all runs no longer breaks the information out by AFSC or SOC. This output table can be found in Appendix B.

The 54 generated schedules must be evaluated. Since the data used in the analysis is partially fictional, the most important information that can be gleaned from these results are the different effects that changes in the parameters make (both the direction

and magnitude of the effects). The following section discusses the 54 different scenarios and their corresponding schedules.

Actual Output Discussion

MIN TDD Versus DIST DD

The first comparison is between the different objective functions: minimizing the total number of down days for all students (MIN TDD) and minimizing the squared number of down days (or more equally distributing the down days) for all students (DIST DD). Using the Dist DD method, the schedule obtained may not be optimal with respect to total number of down days. As shown in Figure 7, the solutions are slightly worse in this case.

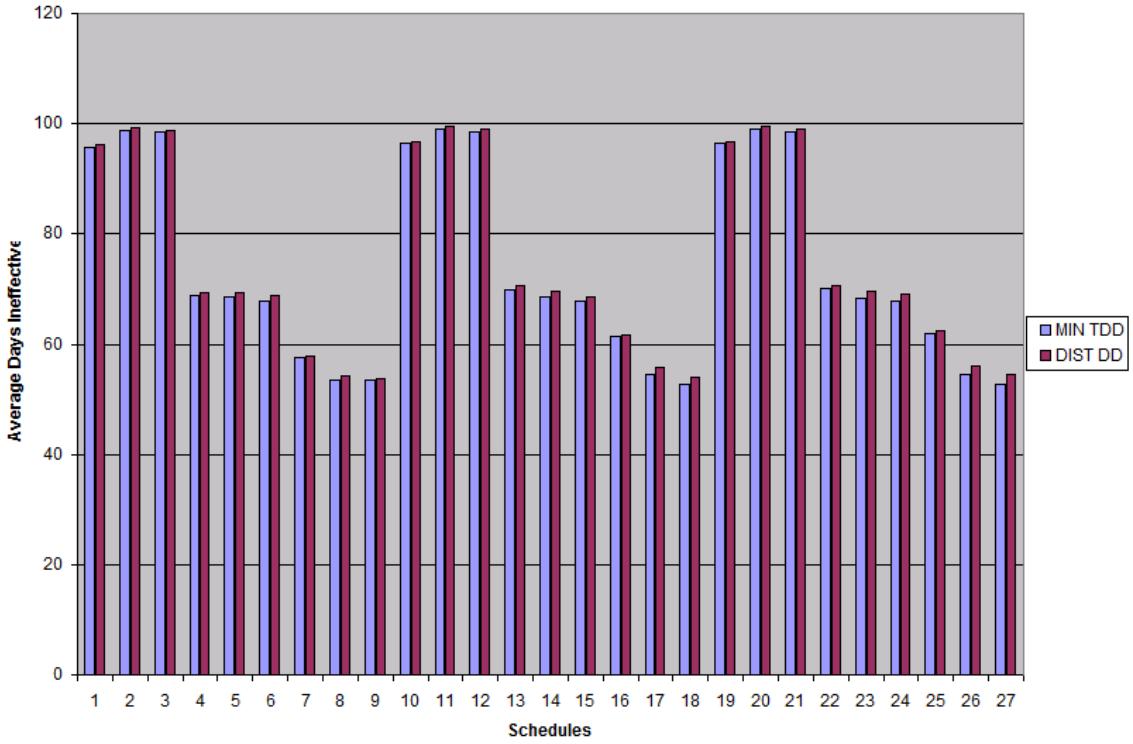

Figure 7: MIN TDD versus DIST DD by Average Days Ineffective

As is expected, the MIN TDD method for computing number of down days is always the same or better than the DIST DD method. In the scenarios where the average

number of down days is equivalent for both methods, the schedule produced by the DIST DD method is preferred. This is because this method distributes the number of down days across all students instead of unfairly penalizing a few. The follow discussion demonstrates the impact that using the Dist DD objective function computation has on the worst schedules generated.

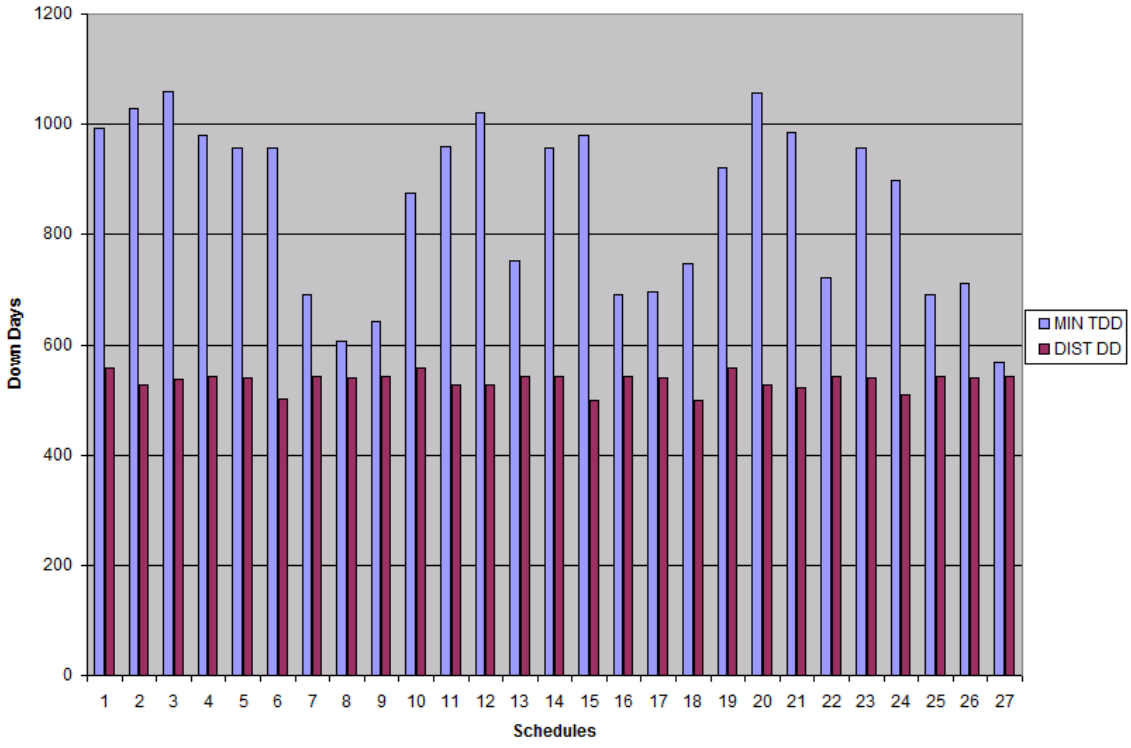

Figure 8: Maximum Number of Down Days for an Individual

As shown in Figure 8, the difference in the worst schedule produced between the two objective function computations is drastic; the second objective function computation method is always superior. Further, the largest rise in the average down days as a result of squaring the arc weights is only 3.41%. These differences indicate that the Dist DD method may be preferred.

The next consideration is the affect that delaying the commissioning of ROTC graduates has on the total number of down days. The biggest benefit of increasing the commissioning from 0 (i.e. immediately upon graduation) to 180 or 365 days is that the

average number of down days decreases. Figure 9 shows the average number of down days given the different inputs for ROTC delay. The purple bar represents the average days spent waiting while the yellow bar represents the average delay an ROTC graduate would experience.

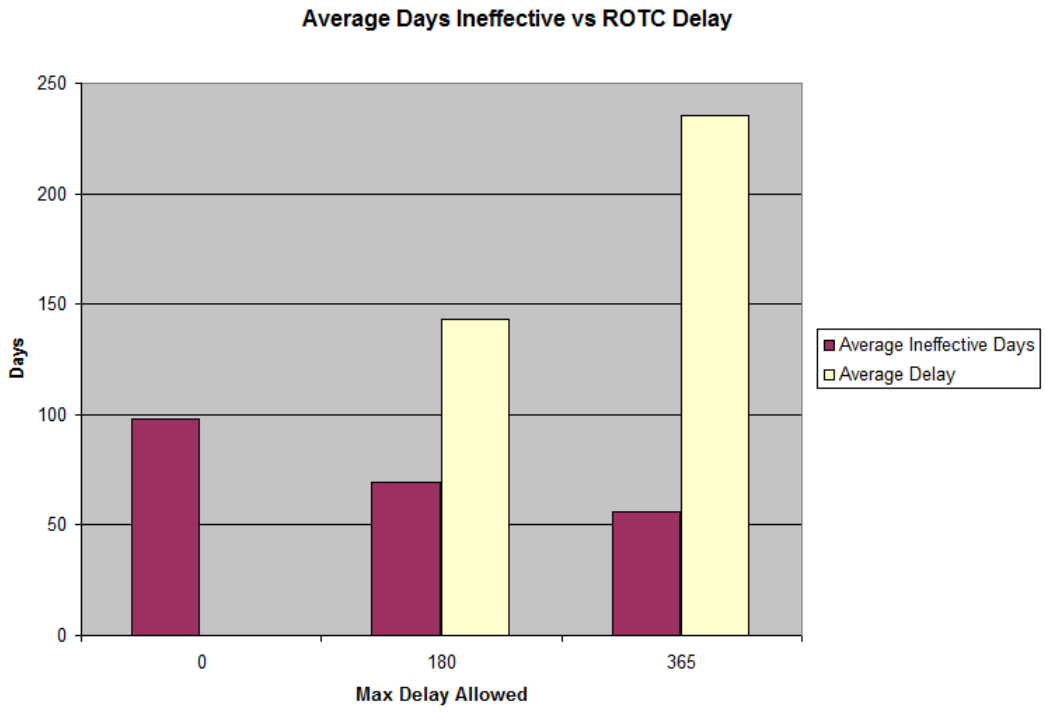

Figure 9: Average Ineffective Days versus ROTC Delay

First, consider the case where no delay is allowed. The average number of down days (or ineffective days) is 98, and the number of days spent waiting for commissioning is 0. When the allowed delay is increased to 180, the average number of ineffective days decreases 69 (a total decrease of 29). The average number of days until commissioning increases to almost 143.

The next change is not quite as significant. By allowing up to a 365 day delay to commissioning for ROTC graduates, the average number of down days decreases to 56 days (a decrease of 13 over the scenario with 180 days delay or 42 over the scenario with no delay). The average number of days delayed for commissioning increases to 235 (an

increase of 92). In order to understand the magnitude of the drops, a conversion to dollars is beneficial.

As discussed earlier, 2nd lieutenants earn approximately $124 per day and approximately 4000 officers are commissioned annually. Therefore, the reduction of 29 down days is equal to approximately $14.4 million in salaries saved. The reduction of the additional 13 down days is about $6.5 million. The Air Force senior leadership must decide if an average of 142 days delayed until commissioning for an ROTC graduate is worth an annual average of $14.4 million to the Air Force. If this is acceptable, they must decide of another 92 days of delay is worth an additional $6.5 million in savings.

The next area of consideration is the different levels of leave allowed for USAFA graduates before they are available to begin their specific IST sequence. This has the effect of allowing USAFA graduates to attend one or more offerings of an ASBC class that they normally would not be able to attend. The amount of leave allowed depends upon the number of ASBC classes offered during the 60 day time frame immediately after graduation (the time normally reserved for USAFA graduate leave). In the data sets used in this research, a single additional ASBC class was offered. The start date for this ASBC class was 33 days after the USAFA graduation date. Thus, USAFA graduates assigned to this course would get 25 days of leave (8 days of travel are required per AFPC and AETC requirements) while the other USAFA graduates received 60 days of leave and attended a later offering of ASBC. Figure 10 shows this information graphically.

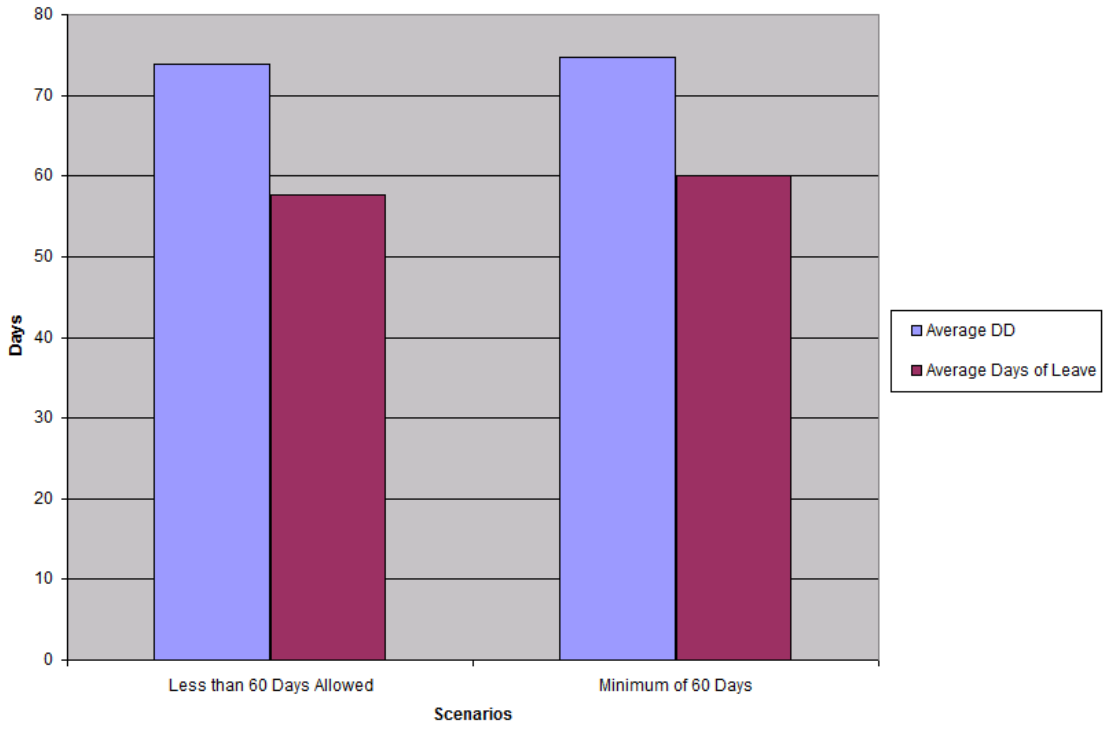

Figure 10: Average Days Ineffective versus USAFA Leave

The left pair of bars is the averages among the runs where USAFA graduates have less than 60 days of leave. The right pair includes the scenarios where a minimum of 60 days of leave was required for USAFA graduates. The average number assigned to the earlier class was 112 and the average USAFA graduate received 57.5 days of leave. However, the overall effect on the average number of down days is minimal. The decrease in average number of down days (for *all* students) is approximately 0.86 days while the average days of leave for USAFA graduates decreases by 2.5 days. This savings of 0.86 days equates to approximately $426,560 in salaries saved. The potential political backlash of forcing USAFA graduates to waive their rights to 60 days of leave may prevent the implementation of this policy. This leave is allocated by congressionally mandated law; commanders cannot force USAFA graduates to waive their rights to 60 days of leave. However, if USAFA graduates are given the option to voluntarily reduce their allotted leave in order to attend an early ASBC class (and thereby be scheduled to an

early IFT and UPT class), this policy could potentially be implemented without negative repercussions.

The final variable under analysis is the different blend levels. In all classes there is a blend level for SOC. ASBC has the additional constraint of blending the various AFSCs in conjunction with the SOC. The outputs of interest in this category are the average number of days of ineffective training and the average number of people forced to not complete training due to insufficient capacity in class offerings. Since two different units (days and people) are simultaneously under consideration, a normalized measure is necessary to view them on the same graph. The least constrained scenario occurs when the maximum allowed of any type officer into any class is 100% of the seats (essentially, no blending is forced). Therefore, this will be the baseline from which it is possible to measure a percent change from the optimal and compare the results. Figure 10 shows this information graphically. The forced 75% blend shows a slight decrease in both measures from the optimal "No Forced Blend." The additional number of officers forced to skip is 11 and the additional average ineffective days is .85 days. These changes seem minor, however, the difference for the 50% blend is much more pronounced. The increase in the number of officers forced to skip is 145 and the increase of the average number of days spent waiting is about 1.9 days. The affect on the additional officers forced to skip is likely unacceptable.

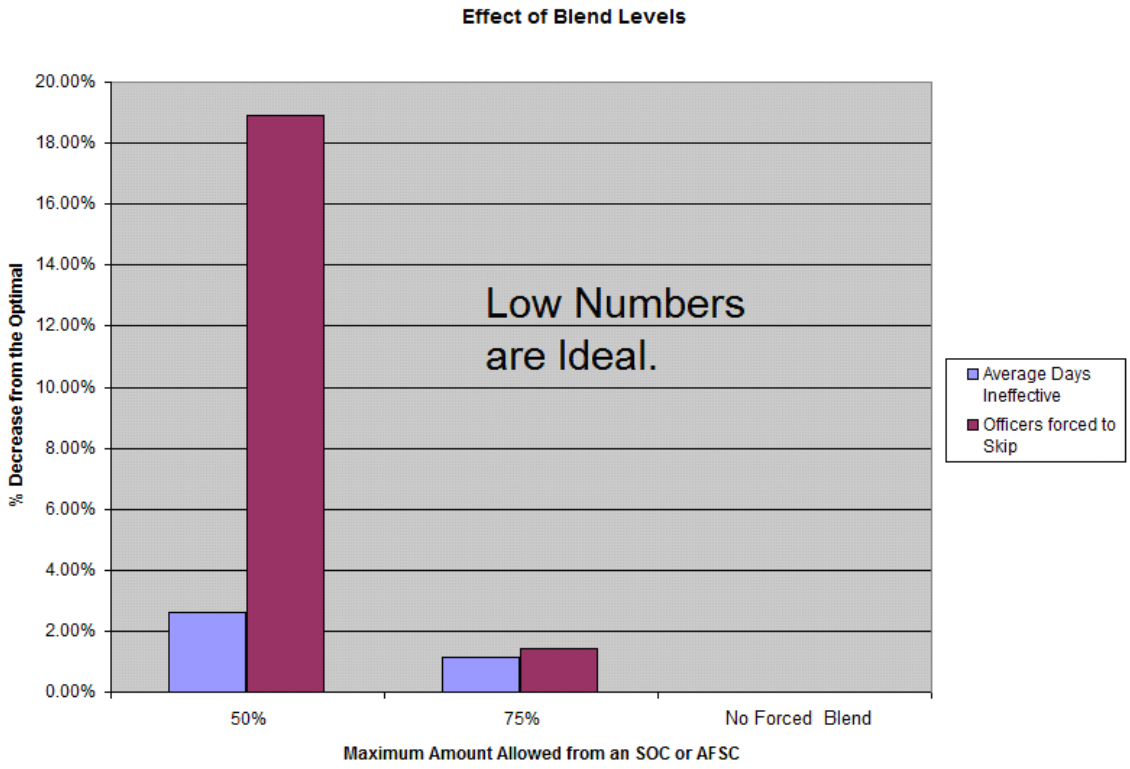

Figure 11: Effect of Blend Levels

Because they produced undesirable results (by forcing a much larger number of students to not complete their training), the scenarios with an upper limit of 50% will no longer be considered; the only scenarios which are acceptable are those with an upper bound of 75% or not forcing any blend at all. The reason the unblended option may be considered is that forcing a blend may not be necessary. Because of the distribution of training courses throughout the year, the optimal schedule using no forced blend creates a schedule that has blends within the courses. For the following comparison, the minimum USAFA leave is set at 60, the maximum ROTC delay is set at 365, the DIST DD objective function computation method is used and the blend situations considered at no blend and 75%. Figure 11 shows the course composition by SOC for ASBC courses with a 75% blend. Figure 12 is the same situation broken down by AFSC instead of SOC.

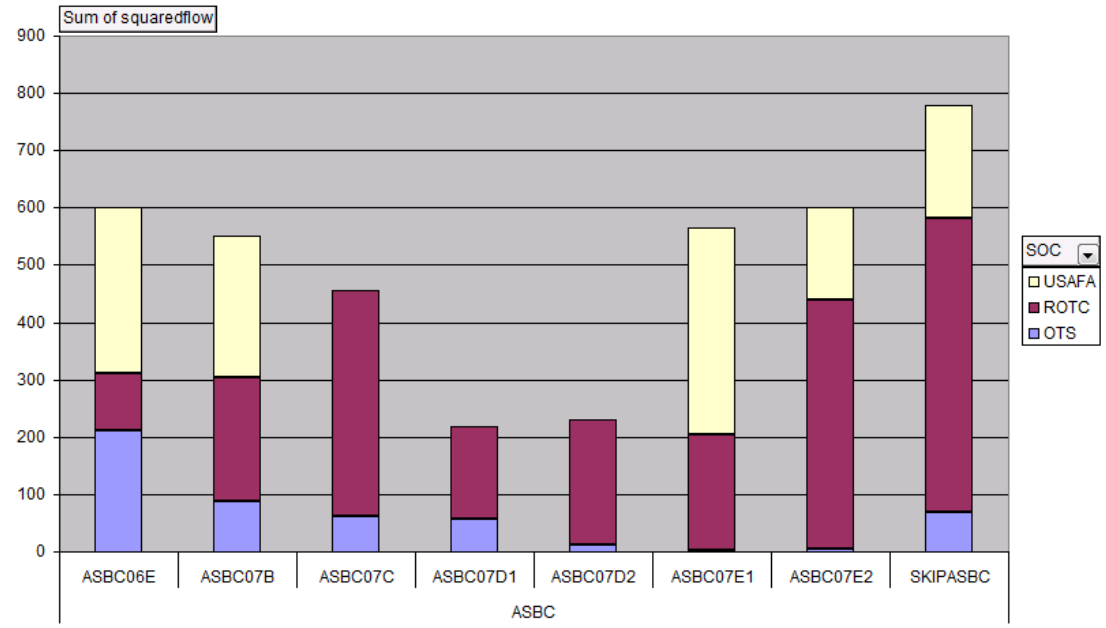

Figure 12: Example ASBC Course Composition by SOC With 75% Blend

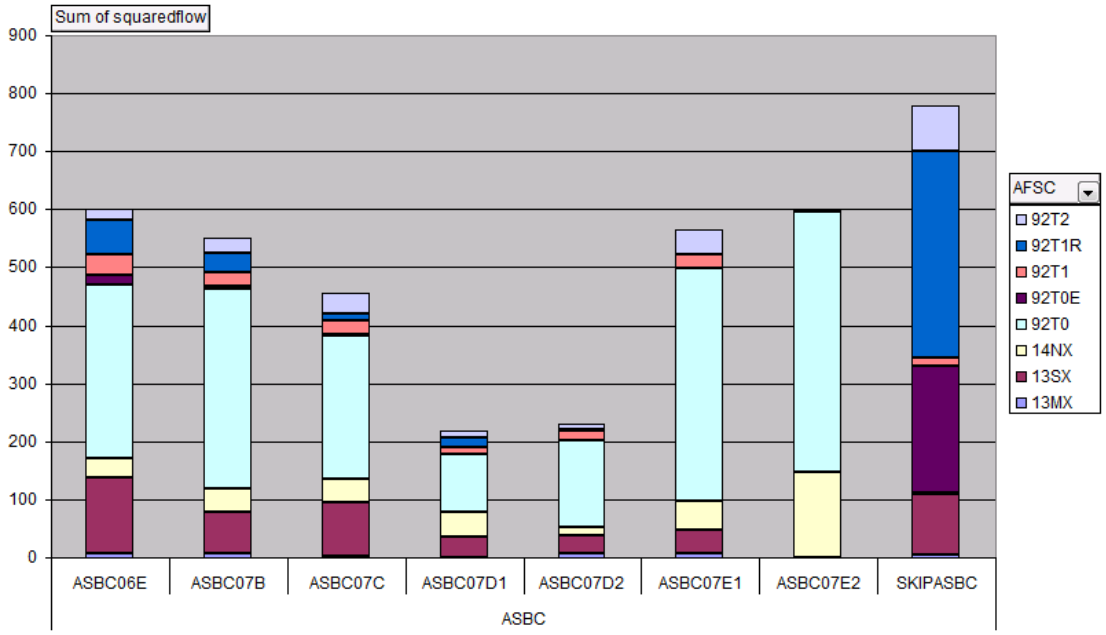

Figure 13: Example ASBC Course Composition by AFSC with 75% Blend

The following figures will show the ASBC classes with the same inputs except there will be no forced blend.

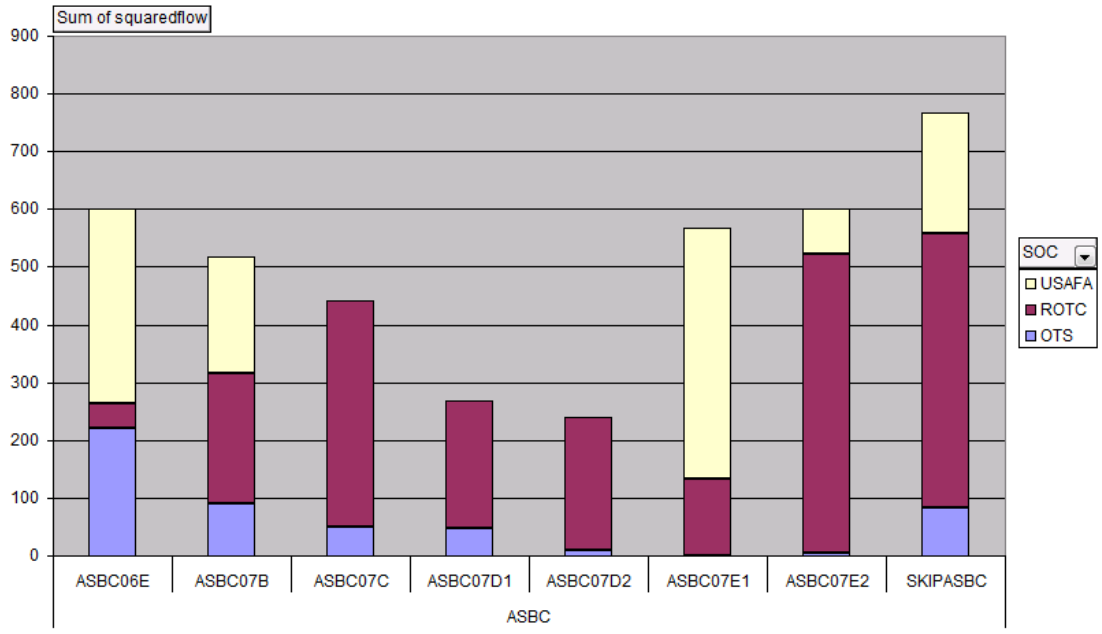

Figure 14: Example ASBC Course Composition by SOC With No Forced Blend

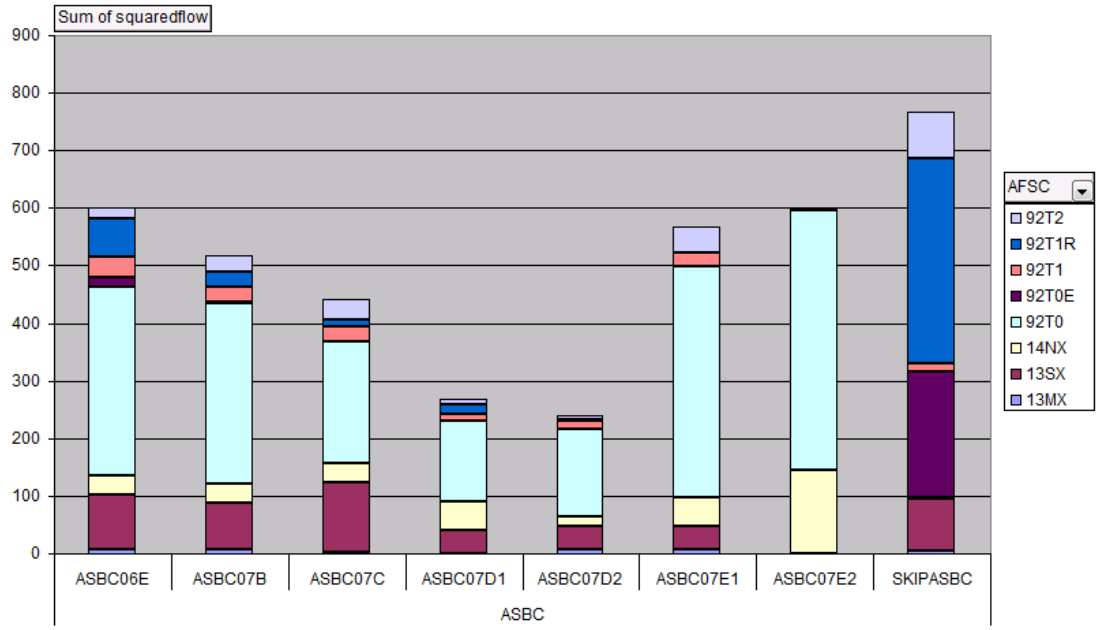

Figure 15: Example ASBC Course Composition by AFSC With No Forced Blend

Using this data set, imposing the 75% blend has little effect on the actual blends within the courses. For this reason, it may be best to simply ignore forced blends and let the optimization run. The result may lead to blended courses. However, if the capacities

of the classes are increased to ensure proper training completion of all students, then forcing the blends may be required.

Summary

This chapter presented results which were generated by NFISTSP on 54 problem scenarios. The analysis of these scenarios allows decision makers to determine which settings are preferable for different parameters. The results from these scenarios also demonstrate the efficiency and effectiveness of the NFISTSP algorithm.

V. Conclusions

Overview

This section outlines the conclusions drawn from the results in the previous chapter. First, recommendations for AETC and AFPC regarding the NFISTSP are presented. Next, potential implementation obstacles are discussed, followed by a discussion of the problem from a mathematical robustness standpoint. Finally, areas for expansion of this research are presented.

Problem Specific Recommendations

Because the specific start and end dates for class offerings of a course vary from year to year, it would be best to perform this entire analysis each time the program must be executed. Unfortunately, due to time constraints, this is not always possible. Some problem generalities can be made which are applicable regardless of these start and end dates.

First, although desirable in the actual class composition, specifying minimal blend levels is not necessary. Through scheduling an optimal schedule, course blends will automatically occur. If it is absolutely necessary to have a specific blend, then specifying it in the formulation is necessary. However, it is likely not necessary to induce a maximum blend percentage on the classes to achieve some level of blending.

With regard to USAFA leave, the benefit of reducing it below the current value of 60 is minimal. This is due to the fact that this only allows for attendance of 1 (or maybe 2) ASBC class offerings at the beginning of the scheduling year. While this can be helpful, the real bottlenecks occur later in training schedule. If it is possible, increasing the class capacity in the courses with small capacities would have a greater effect.

Delaying ROTC commissioning is likely a sensitive issue with potential to severe political backlash. Unfortunately, these personnel will not be receiving a paycheck or

covered by the military health and life insurance during this period. The analysis demonstrates that there are potentials for improvements in the schedule by incorporating a maximum of 180 days delay in commissioning. This occurs largely because it allows the schedule to delay these people until the portion of the scheduling year when USAFA and OTS are not graduating people. However, the strain put on ROTC graduates may preclude the AF senior leadership from actually incorporating this policy. The small improvements in the schedule which result from delaying the commissioning of ROTC students 365 days does not outweigh the drawbacks and should not be considered in future analysis.

Finally, with regard to the optimal objective function calculation method, the method of distributing the total number of down days (DIST DD) across the overall student population is preferred to that of simply minimizing the total number of down days (MIN TDD). The average number of down days does not increase greatly between the two methods, and the DIST DD method greatly reduces the worst schedule while minimally affecting the average performance.

At any of the settings, there are a few guarantees of all the schedules produced. Every schedule will be optimal for its given inputs. Every schedule will blend the courses to some level of satisfaction (although this level may be required to be any value). Every schedule will not assign someone to take a course without being scheduled for all its prerequisites.

Problems to Overcome

The NFISTSP addresses the desired issues but not without an associated cost. In order to use this solution method effectively, the inputs need to include ample classes for each course. Ideally, the course inputs should include sufficient class capacity to support all students which require the course. NFISTSP could then generate a schedule in which

no course skipping occurs. In the sample data available, this was not the case. However, through some organizational planning, this should not be a difficult feat to achieve.

Additional Robustness

This approach is very robust. In this formulation, the people to be scheduled are considered heterogeneous (different in some attributes). In this case, the people had different AFSCs and SOCs. However, this formulation would allow for any number of heterogeneous traits and any number of types within those traits.

Additionally, there is no limit to the time horizon that NFISTSP could accommodate. As long as the courses and students to be scheduled have corresponding calendar years, the NFISTSP will schedule decades of students. It could also be used to forecast schedules using projected (as opposed to actual) dates and values. This could be extremely beneficial for determining the actual start and end dates for a course, as well as the capacity for each class offering.

Future Research

There are many areas for future research, both in the specific and general formulation as well as the NFISTSP algorithm itself. First, in the specific problem, the model could be modified to be given a set of officers that need training and output the optimal class dates and sizes. This would require even more lead time in the planning process but could save the Air Force more money by reducing the number of instructors needed for smaller classes. It would reduce wait times between courses as well as reducing the number of empty seats in a class. Also, the algorithm could easily be expanded to incorporate all career fields.

In the area of the NFISTSP computer code, there is room for improvement. SAS has recently released a new SAS/OR optimization tool package that is closer in syntax to CPLEX and other competitive solvers. The SAS/OR syntax used by this thesis was

based on an older version of SAS/OR. The recent improvements would make the computer code more easily readable for those not extremely familiar with the SAS/OR language.

In the area of the mathematics and formulation there are additional areas of research as well. First is the use of Decision Analysis (DA) as a method for determining schedules with multiple (and sometimes conflicting) goals. It is clear that the decision maker is trying to balance many different measures in the schedule (average ineffective days, blends within classes, average delay until ROTC commissioning, etc). It seems very suiting that DA could be used to select that best schedule, rather than having a decision maker attempt this alone. Further, the value model developed by the DA could be incorporated into the SAS code and used to produce the schedules creating better schedules.

Another potential area of expansion for this research is the introduction of probabilities and queuing theory. With some probability, every student might actual fail a course; these attrition rates and probabilities could be incorporated in the NFISTSP as well.

Appendix A: SAS Code

```
/*****************************************************************/
/*                                                               */
/*              IMPORT ALL DATA FILES FROM EXCEL                 */
/*                                                               */
/*                                                               */
/*                                                               */
/*****************************************************************/

/* IMPORT FILES: ASBC COURSE LIST, IST COURSE LIST, IFS COURSE LIST,
PEOPLE LIST, AFSC LIST, COMMISSIONING SOURCE LIST */
PROC IMPORT OUT=ASBC
      DATAFILE="C:\Illig\SAS\Thesis\christina\ASBC_all.XLS"
      DBMS=EXCEL2000 REPLACE;
RUN;
PROC IMPORT OUT=IST
      DATAFILE="C:\Illig\SAS\Thesis\christina\IST_all.XLS"
      DBMS=EXCEL2000 REPLACE;
RUN;
PROC IMPORT OUT=IFS
      DATAFILE="C:\Illig\SAS\Thesis\christina\IFS_all.XLS"
      DBMS=EXCEL2000 REPLACE;
RUN;
PROC IMPORT OUT=PEOPLE
      DATAFILE="C:\Illig\SAS\Thesis\christina\PEOPLE.XLS"
      DBMS=EXCEL2000 REPLACE;
RUN;
PROC IMPORT OUT=AFSC
      DATAFILE="C:\Illig\SAS\Thesis\christina\AFSC.XLS"
      DBMS=EXCEL2000 REPLACE;
RUN;
PROC IMPORT OUT=SOC
      DATAFILE="C:\Illig\SAS\Thesis\christina\SOC.XLS"
      DBMS=EXCEL2000 REPLACE;
RUN;


/*****************************************************************/
/*                                                               */
/*              FORMAT THE 'PEOPLE' FILE                         */
/*                                                               */
/*                                                               */
/*                                                               */
/*****************************************************************/

/* FORMAT: INITIAL FORMATTING--SSN, RNLTD, COURSE INFO, ETC */
DATA People_Formatted(keep=SSN AFSC SOC DOC READY COUNT);

      set People;

      format READY date.;

      /*first available date*/
      IF SOC='USAFA' THEN READY=DOC+60;
      ELSE READY=DOC;

      /*Create count variable*/
```

```
      COUNT=1;

      /* FORMAT AFSC: PLACE ALL AFSC IN 4 DIGIT FORMAT WITH X FOR
QUALIFICATION LEVEL W/EXCEPTION OF NAVS, 33SXA */
      IF AFSC IN ('92T0' '92T0N' '92T2' '92T2X') THEN
AFSC=SUBSTR(AFSC,1,4);
      ELSE IF AFSC = '31S1A' THEN AFSC='33SXA';
      ELSE IF AFSC IN ('13D1A' '13D1B' '13S1X' '14N1X' '21M1S' '21M1V')
THEN AFSC=SUBSTR(AFSC,1,3) || 'X';
      ELSE IF SUBSTR(AFSC,1,5)='X' THEN AFSC = SUBSTR(AFSC,1,4);
      ELSE IF AFSC IN ('92T1R') THEN AFSC = '92T1R';   /* SPECIAL CASE
FOR NAVS AT RANDOLPH--IDENTIFIER */
      ELSE IF AFSC IN ('92T1' '92T1 ') THEN AFSC = '92T1';     /*
SPECIAL CASE FOR NAVS AT PENSACOLA--IDENTIFIER */
      ELSE IF AFSC IN ('92T0E') THEN AFSC = '92T0E'; /* SPECIAL CASE
FOR PILOTS SELECTED FOR ENJEPT */
      ELSE AFSC = SUBSTR(AFSC, 1, 3) || 'X';

      /*only consider the 6 critical AFSCs*/
      IF AFSC NOT IN ('13MX' '13SX' '14NX' '92T0' '92T1' '92T1R' '92T2'
'92T0E') THEN DELETE;

RUN;
/* CHECK FOR DUPLICATES WITHIN THE FILE */
PROC SORT DATA=PEOPLE_formatted; BY SSN; RUN;
DATA PEOPLE_formatted2;
      SET PEOPLE_formatted;
      BY SSN;
      IF FIRST.SSN;
RUN;

/*****************************************************************/
/*                                                               */
/*        CREATE THE SOURCE NODES FOR THE NETFLOW COMMAND        */
/*                                                               */
/*                                                               */
/*                                                               */
/*****************************************************************/

/*sort new people file by time available, commissioning source and
AFSC*/
PROC SORT DATA=People_formatted2; BY READY DOC SOC AFSC; RUN;

/*Create list of date available, commissiong source and AFSC
combinations and number of people for each set*/
PROC SUMMARY DATA=People_formatted2;
      by READY DOC SOC AFSC;
      freq Count;
      output  out=NODES_Source;
run;

/*delete the _type_ variable in each table and put in the BASE
variable*/
DATA NODES_Source;
      set NODES_Source(drop=_type_);

      /*date to start counting time at*/
      IF SOC='ROTC' THEN BASE=DOC+180;
      IF SOC='USAFA' THEN BASE=DOC+60;
```

```
      IF SOC='OTS' THEN BASE=DOC;

      format BASE date.;

RUN;

DATA NODES_Source;
      set NODES_Source;

      /*supply amount at node*/
      _supdem_=_freq_;

      /*create unique names for AFSC, SOC and the name of the node*/
      _node_=compress(left(trim(SOC)||'_'||trim(AFSC)||'_'||trim(READY)
));
      source_afsc=afsc;
      source_soc=soc;
RUN;

/*create a list of nodes for each afsc and SOC combination (amount = 0)
so that everyone is accounted for*/
/*in order to create the nodes, sort all the lists involved*/
PROC SORT DATA=SOC; BY SOC; RUN;
PROC SORT DATA=AFSC; BY AFSC; RUN;

/*create the possible combinations of AFSC and SOC*/
PROC SQL;
            CREATE TABLE Source_Base AS
            SELECT  B.SOC,
                        C.AFSC
                  FROM SOC AS B, AFSC AS C;
QUIT;

Data source_base;
      set source_base;
      ready=10000;
      _freq_=0;
      base=10000;
      _supdem_=0;
      source_afsc=afsc;
      source_soc=soc;
      _node_=compress(left(trim(SOC)||'_'||trim(AFSC)||'_'||trim(READY)
));
RUN;

/*put base source info onto the current existing info*/
DATA Nodes_Source;
      set Nodes_Source
            source_base;
RUN;



/*make the node list used in netflow command (not referenced elsewhere
just here)*/
DATA _Node_(keep= _node_ _supdem_);
      set Nodes_source;
RUN;
```

```
/*****************************************************************/
/*                                                               */
/*        CREATE THE ASBC NODES FOR THE NETFLOW COMMAND          */
/*                                                               */
/*                                                               */
/*                                                               */
/*****************************************************************/

/*in order to create the nodes, sort all the lists involved*/
PROC SORT DATA=ASBC; BY ASBC_COURSE; RUN;
PROC SORT DATA=SOC; BY SOC; RUN;
PROC SORT DATA=AFSC; BY AFSC; RUN;

/*create the possible combinations of AFSC and SOC for each ASBC
class*/
PROC SQL;
            CREATE TABLE Nodes_ASBC AS
            SELECT  A.ASBC_COURSE,
                        A.ASBC_CSD_,
                        A.ASBC_CGD_,
                        A.ASBC_SEATS,
                        B.SOC,
                        B.minpercentasbcbysource,
                        B.maxpercentasbcbysource,
                        C.AFSC,
                        C.minpercentasbcbyafsc,
                        C.maxpercentasbcbyafsc
                  FROM ASBC AS A, SOC AS B, AFSC AS C;
QUIT;

DATA Nodes_ASBC ;
      set Nodes_ASBC;

      /*put in form like others*/
      ASBC_CSD=ASBC_CSD_;
      ASBC_CGD=ASBC_CGD_;

      /*find the minimum and maximum amounts (not percents) by both SOC
and AFSC*/
      minsoc=floor(ASBC_seats*minpercentasbcbysource/100);
      maxsoc=ceil(ASBC_seats*maxpercentasbcbysource/100);
      minafsc=floor(ASBC_seats*minpercentasbcbyafsc/100);
      maxafsc=ceil(ASBC_seats*maxpercentasbcbyafsc/100);
RUN;


/*create node so people can go unassigned to ASBC (shows that not
enough ASBC classes are available or contraints are too tight)*/
PROC SQL;
            CREATE TABLE SKIP_ASBC AS
            SELECT  A.SOC,
                        B.AFSC
            FROM SOC AS A, AFSC AS B;
QUIT;

DATA SKIP_ASBC ;
      set SKIP_ASBC;

      Asbc_course='SKIPASBC';
```

```
      minsoc=0;
      minafsc=0;
      maxsoc=4000;
      maxafsc=4000;

      format ASBC_CSD DATE9.;

      ASBC_CSD=30000;
      ASBC_CGD=.;

RUN;

/*join source and skip nodes*/
PROC APPEND BASE=Nodes_ASBC DATA=skip_ASBC; RUN;

/*rename the SOC and AFSC so they are unique in this table and clean
the table up*/
DATA Nodes_asbc;

      set Nodes_asbc;

      /*unique names*/
      asbc_afsc=afsc;
      asbc_soc=soc;
      asbc_node=compress(trim(SOC)||'_'||trim(AFSC)||'_'||trim(ASBC_Cou
rse));



RUN;
/*done creating ASBC Nodes*/

/*****************************************************************/
/*                                                               */
/*        CREATE THE IFS NODES FOR THE NETFLOW COMMAND           */
/*                                                               */
/*                                                               */
/*                                                               */
/*****************************************************************/

/*create list of pilot afscs (just 92T0 and 92T0E)*/
DATA afsc_pilot;
      set afsc;
      if afsc not in ('92T0' '92T0E') then delete;
RUN;

/*in order to create the nodes, sort all the lists involved*/
PROC SORT DATA=IFS; BY IFS_COURSE; RUN;
PROC SORT DATA=SOC; BY SOC; RUN;
PROC SORT DATA=AFSC_pilot; BY AFSC; RUN;

/*create all nodes for IFS (need to track AFSC becuase 2 are going
here: 92T0 and 92T0E)*/
PROC SQL;
            CREATE TABLE Nodes_IFS AS
            SELECT A.IFS_COURSE,
                        A.IFS_CSD,
                        A.IFS_CGD,
```

```
                        A.IFS_SEATS,
                        B.SOC,
                        B.minpercentifsbysource,
                        B.maxpercentifsbysource,
                        C.AFSC
                    FROM IFS AS A, SOC AS B, AFSC_pilot AS C;
QUIT;

DATA Nodes_IFS ;
      set Nodes_IFS;
      /*find the minimum and maximum amounts (not percents) by SOC*/
      minsoc=floor(IFS_seats*minpercentifsbysource/100);
      maxsoc=ceil(IFS_seats*maxpercentifsbysource/100);
RUN;


/*create node so people can go unassigned to IFS (shows that not enough
IFS classes are available or constraints are too tight)*/
PROC SQL;
            CREATE TABLE SKIP_IFS AS
            SELECT  A.SOC,
                        B.AFSC
            FROM SOC AS A, AFSC_pilot AS B;
QUIT;

DATA SKIP_IFS ;
      set SKIP_IFS;
      ifs_course='SKIPIFS';

      minsoc=0;
      maxsoc=4000;

      format IFS_CSD DATE9.;

      IFS_CSD=30000;
      IFS_CGD=.;
RUN;

/*join source and skip nodes*/
Data Nodes_ifs;
      format IFS_Course $Char7.;
      set nodes_ifs
            skip_ifs;
RUN;

PROC APPEND BASE=Nodes_ifs DATA=skip_ifs; RUN;

/*rename the SOC and AFSC so they are unique in this table and clean
the table up*/
DATA Nodes_ifs;
      set Nodes_ifs;

      /*Create a unique name for the node and make AFSC and SOC columns
unique*/
      ifs_node=compress(trim(SOC)||'_'||trim(AFSC)||'_'||trim(IFS_COURS
E));
      ifs_afsc=afsc;
      ifs_soc=soc;
```

```
RUN;

/*done creating IFS nodes.*/

/*****************************************************************/
/*                                                               */
/*        CREATE THE IST NODES FOR THE NETFLOW COMMAND           */
/*                                                               */
/*                                                               */
/*                                                               */
/*****************************************************************/

/*list classes for non-pilots*/
DATA IST_nonpilot;
      set IST;
      IF IST_AFSC in ('92T0' '92T0E') THEN DELETE;
RUN;

/*list afscs for non-pilots*/
DATA afsc_nonpilot;
      set AFSC;
      IF AFSC in ('92T0' '92T0E') THEN DELETE;
RUN;

/*in order to create the nodes, sort all the lists involved*/
PROC SORT DATA=IST_nonpilot; BY IST_COURSE; RUN;
PROC SORT DATA=SOC; BY SOC; RUN;

/*create all nodes for IST*/
PROC SQL;
            CREATE TABLE Nodes_IST AS
            SELECT A.IST_COURSE,
                        A.IST_CSD,
                        A.IST_CGD,
                        A.IST_SEATS,
                        A.IST_AFSC,
                        B.SOC,
                        B.minpercentistbysource,
                        B.maxpercentistbysource
                  FROM IST_nonpilot AS A, SOC AS B;
QUIT;

DATA Nodes_IST;
      set Nodes_IST;

      afsc=ist_afsc;

      /*find the minimum and maximum amounts (not percents) by SOC*/
      minsoc=floor(ist_seats*minpercentistbysource/100);
      maxsoc=ceil(ist_seats*maxpercentistbysource/100);
RUN;


/*create nodes so people can go unassigned to IST (shows that not
enough IST classes are available or constraints are too tight)*/
PROC SQL;
           CREATE TABLE SKIP_IST AS
           SELECT  A.SOC,
                       B.AFSC
```

```
            FROM SOC AS A, AFSC_nonpilot AS B;
QUIT;


DATA SKIP_IST;
      set SKIP_IST;
      ist_course='SKIPIST';
      minsoc=0;
      maxsoc=100000;

      format IST_CSD DATE9.;

      IST_CSD=30000;

      IST_AFSC=AFSC;
RUN;

/*join source and skip nodes*/
PROC APPEND BASE=Nodes_IST DATA=skip_IST; RUN;

/*rename the SOC so it is unique in this table*/
DATA Nodes_IST;
      set Nodes_IST;

      /*unique names*/
      ist_node=compress(trim(SOC)||'_'||trim(IST_AFSC)||'_'||trim(ist_C
ourse));
      IST_soc=soc;
RUN;

/*done creating the IST nodes.*/

/*****************************************************************/
/*                                                               */
/*        CREATE THE UPT NODES FOR THE NETFLOW COMMAND           */
/*                                                               */
/*                                                               */
/*                                                               */
/*****************************************************************/

/*make the list of UPT courses*/
DATA UPT(drop=IST_course IST_seats IST_CSD IST_CGD IST_AFSC);
      set IST;
      IF IST_AFSC not in ('92T0' '92T0E') THEN DELETE;
      UPT_Course=IST_course;
      UPT_seats=IST_seats;
      UPT_CSD=IST_CSD;
      UPT_CGD=IST_CGD;
      UPT_AFSC=IST_AFSC;
RUN;

/*in order to create the nodes, sort all the lists involved*/
PROC SORT DATA=UPT; BY UPT_COURSE; RUN;
PROC SORT DATA=SOC; BY SOC; RUN;

/*create all nodes for UPT*/
PROC SQL;
            CREATE TABLE Nodes_UPT AS
            SELECT A.UPT_COURSE,
```

```
                        A.UPT_CSD,
                        A.UPT_CGD,
                        A.UPT_SEATS,
                        A.UPT_AFSC,
                        B.SOC,
                        B.minpercentuptbysource,
                        B.maxpercentuptbysource
                    FROM UPT AS A, SOC AS B;
QUIT;

DATA Nodes_UPT ;
      set Nodes_UPT;

      /*find the minimum and maximum amounts (not percents) by SOC*/
      minsoc=floor(UPT_seats*minpercentuptbysource/100);
      maxsoc=ceil(UPT_seats*maxpercentuptbysource/100);

      afsc=UPT_afsc;

RUN;

/*create nodes so people can go unassigned to UPT (shows that not
enough UPT classes are available or constraints are too tight)*/
PROC SQL;
            CREATE TABLE SKIP_UPT AS
            SELECT  A.SOC,
                        B.AFSC
            FROM SOC AS A, AFSC_pilot AS B;
QUIT;

DATA SKIP_UPT;
      set SKIP_UPT;
      UPT_Course='SKIPUPT';

      minsoc=0;
      maxsoc=100000;

      UPT_AFSC=afsc;

      format UPT_CSD DATE9.;

      UPT_CSD=30000;
      UPT_CGD=.;
RUN;

/*join source and skip nodes*/
PROC APPEND BASE=Nodes_UPT DATA=skip_UPT; RUN;

DATA Nodes_UPT;
      set Nodes_UPT;

      UPT_SOC=soc;

      /*Create a unique name for the node and soc*/
      UPT_node=compress(trim(UPT_SOC)||'_'||trim(UPT_AFSC)||'_'||trim(U
PT_COURSE));

RUN;
```

```
/*done creating UPT nodes.*/
/*done creating ALL nodes*/

/*****************************************************************/
/*                                                               */
/*    CREATE THE ASBC ARCS FOR THE NETFLOW COMMAND               */
/*                                                               */
/*                                                               */
/*                                                               */
/*****************************************************************/


/* first, sort all the lists involved*/
PROC SORT DATA=Nodes_Source; BY _node_; RUN;
PROC SORT DATA=Nodes_asbc; BY asbc_node; RUN;

PROC SQL;
            CREATE TABLE ARCS_SOURCE_TO_ASBC AS
            SELECT  A._node_,
                        A._supdem_,
                        A.source_AFSC,
                        A.source_SOC,
                        A.READY,
                        A.DOC,
                        A.Base,
                        B.asbc_node,
                        B.asbc_course,
                        B.asbc_AFSC,
                        B.asbc_SOC,
                        B.ASBC_CSD,
                        B.ASBC_CGD,
                        B.minsoc,
                        B.minafsc,
                        B.maxsoc,
                        B.maxAFSC
                  FROM Nodes_Source AS A, Nodes_asbc AS B;
QUIT;

/*now trim out arcs that cannot exist*/
DATA ARCS_SOURCE_TO_ASBC;
      set ARCS_SOURCE_TO_ASBC;

      /*SOC and AFSC must match (to keep track of blend amounts)*/
      IF source_AFSC ^= asbc_AFSC THEN DELETE;
      IF source_SOC ^= asbc_SOC THEN DELETE;

      /*can't go to a class before you are ready*/
      IF READY > ASBC_CSD THEN DELETE;

      /*possibly consider changing 8 day constraint to allow this
possibility*/
      IF READY + 8 > ASBC_CSD THEN close=1;
      ELSE close=0;

      /*find the cost of sending a person across an arc*/
      _cost_=MAX(0,ASBC_CSD-Base);

      /*create a unique name for the arc*/
```

```
      _name_=compress(trim(source_soc)||'_'||trim(Source_afsc)||'_'||tr
im(ready)||'_'||trim(asbc_course));

      /*create variable to show how much leave a USAFA grad will get if
they go across this arc*/
      if source_soc="USAFA" then USAFA_Leave=min(60, ASBC_CSD-DOC);

      /*create variable to show how long a ROTC grad will wait if they
go across this arc*/
      if source_soc="ROTC" then ROTC_Delay=min(180,ASBC_CSD-DOC);

RUN;
/*finished the arcs to ASBC classes.  now we must make arcs to the next
phase in training*/


/*****************************************************************/
/*                                                               */
/*    CREATE THE ARCS FROM ASBC TO IFS FOR THE NETFLOW COMMAND   */
/*                     (APPLIES TO PILOTS)                       */
/*                                                               */
/*                                                               */
/*****************************************************************/

/*create list of pilot asbc nodes*/
Data Nodes_asbc_pilot;
      set Nodes_asbc;

      /*must be pilot to go to IFS*/
      IF asbc_afsc not in ('92T0' '92T0E') THEN DELETE;

      /*can't go if ASBC was skipped*/
      IF ASBC_CGD= . THEN DELETE;
RUN;

/* first, sort all the lists involved*/
PROC SORT DATA=Nodes_asbc_pilot; BY asbc_node; RUN;
PROC SORT DATA=Nodes_ifs; BY ifs_node; RUN;

PROC SQL;
            CREATE TABLE ARCS_ASBC_TO_IFS AS
            SELECT  A.asbc_node,
                        A.asbc_course,
                        A.asbc_AFSC,
                        A.asbc_SOC,
                        A.ASBC_CGD,
                        B.IFS_node,
                        B.IFS_Course,
                        B.ifs_afsc,
                        B.ifs_soc,
                        B.IFS_CSD,
                        B.IFS_CGD,
                        B.minsoc,
                        B.maxsoc
                  FROM Nodes_asbc_pilot AS A, Nodes_ifs AS B;
QUIT;

/*now trim out arcs that cannot exist*/
DATA ARCS_ASBC_TO_IFS;
```

```
      set ARCS_ASBC_TO_IFS;

      /*SOC and AFSC must match (to keep track of blend amounts)*/
      IF ASBC_AFSC ^= IFS_AFSC THEN DELETE;
      IF ASBC_SOC ^= IFS_SOC THEN DELETE;

      /*can't go to a class before you are ready*/
      IF ASBC_CGD > IFS_CSD THEN DELETE;

      /*possibly consider changing 8 day constraint to allow this
possibility*/
      IF ASBC_CGD + 8 > IFS_CSD THEN close=1;
      ELSE close=0;

      /*find the cost of sending a person across an arc*/
      _cost_=IFS_CSD-ASBC_CGD;

      /*create a unique name for the arc*/
      _name_=compress(trim(ASBC_soc)||'_'||trim(ASBC_afsc)||'_'||trim(A
SBC_Course)||'_'||trim(IFS_course));

RUN;
/*finished the arcs to IFS classes.  now we must make arcs from ASBC to
the IST classes for non-pilots*/

/*****************************************************************/
/*                                                               */
/*    CREATE THE ARCS FROM ASBC TO IST FOR THE NETFLOW COMMAND   */
/*                     (APPLIES TO NON-PILOTS)                   */
/*                                                               */
/*                                                               */
/*****************************************************************/

/*asbc nodes for non-pilots that actaully were assigned an ASBC class*/
DATA Nodes_asbc_nonpilot;
      set nodes_asbc;
      IF ASBC_AFSC in ('92T0' '92T0E') THEN DELETE;

      /*can't go if ASBC was skipped*/
      IF ASBC_CGD= . THEN DELETE;
RUN;


/* first, sort all the lists involved*/
PROC SORT DATA=Nodes_asbc_nonpilot; BY asbc_node; RUN;
PROC SORT DATA=Nodes_ist; BY ist_node; RUN;

PROC SQL;
            CREATE TABLE ARCS_ASBC_TO_IST AS
            SELECT  A.asbc_node,
                        A.asbc_course,
                        A.asbc_AFSC,
                        A.asbc_SOC,
                        A.ASBC_CGD,
                        B.IST_node,
                        B.ist_course,
                        B.IST_afsc,
                        B.IST_soc,
                        B.IST_CSD,
```

```
                        B.IST_CGD,
                        B.minsoc,
                        B.maxsoc
                    FROM Nodes_asbc_nonpilot AS A, Nodes_IST AS B;
QUIT;

/*now trim out arcs that cannot exist*/
DATA ARCS_ASBC_TO_IST;
      set ARCS_ASBC_TO_IST;

      /*SOC and AFSC must match (to keep track of blend amounts)*/
      IF ASBC_AFSC ^= IST_AFSC THEN DELETE;
      IF ASBC_SOC ^= IST_SOC THEN DELETE;

      /*can't go to a class before you are ready*/
      IF ASBC_CGD > IST_CSD THEN DELETE;

      /*possibly consider changing 8 day constraint to allow this
possibility*/
      IF ASBC_CGD + 8 > IST_CSD THEN close=1;
      ELSE close=0;

      /*find the cost of sending a person across and arc*/
      _cost_=IST_CSD-ASBC_CGD;

      /*create a unique name for the arc*/
      _name_=compress(trim(ASBC_soc)||'_'||trim(ASBC_afsc)||'_'||trim(A
SBC_Course)||'_'||trim(IST_course));

RUN;
/*finished the arcs to IST classes.  now we must make arcs from IFT to
the UPT classes for pilots (both 92T0 and 92T0E)*/

/*****************************************************************/
/*                                                               */
/*    CREATE THE ARCS FROM IFS TO UPT FOR THE NETFLOW COMMAND    */
/*                     (APPLIES TO PILOTS)                       */
/*                                                               */
/*                                                               */
/*****************************************************************/



/*only use actual attenders of IFS (not nodes for skipping)*/
DATA Nodes_ifs_noskip;
      set Nodes_ifs;
      IF IFS_CGD=. THEN DELETE;
RUN;

/* first, sort all the lists involved*/
PROC SORT DATA=Nodes_IFS_noskip; BY ifs_node; RUN;
PROC SORT DATA=Nodes_upt; BY upt_node; RUN;

PROC SQL;
            CREATE TABLE ARCS_IFS_TO_UPT AS
            SELECT  A.IFS_node,
                        A.IFS_Course,
                        A.IFS_AFSC,
                        A.IFS_SOC,
```

```
                        A.IFS_CGD,
                        B.UPT_node,
                        B.UPT_COURSE,
                        B.UPT_afsc,
                        B.UPT_soc,
                        B.UPT_CSD,
                        B.UPT_CGD,
                        B.minsoc,
                        B.maxsoc
                    FROM Nodes_IFS_noskip AS A, Nodes_UPT AS B;
QUIT;

/*now trim out arcs that cannot exist*/
DATA ARCS_IFS_TO_UPT;
      set ARCS_IFS_TO_UPT;

      /*SOC and AFSC must match (to keep track of blend amounts)*/
      IF IFS_AFSC ^= UPT_AFSC THEN DELETE;
      IF IFS_SOC ^= UPT_SOC THEN DELETE;

      /*can't go to a class before you graduate the previouse*/
      IF IFS_CGD > UPT_CSD THEN DELETE;

      /*possibly consider changing 8 day constraint to allow this
possibility*/
      IF IFS_CGD + 8 > UPT_CSD THEN close=1;
      ELSE close=0;

      /*find the cost of sending a person across an arc*/
      _cost_=UPT_CSD-IFS_CGD;

      /*create a unique name for the arc*/
      _name_=compress(trim(IFS_soc)||'_'||trim(IFS_afsc)||'_'||trim(IFS
_Course)||'_'||trim(UPT_course));

RUN;
/*finished the arcs to UPT classes*/

/*****************************************************************/
/*                                                               */
/* CREATE THE ARCS FROM UPT, IST AND SKIP NODES TO THE SINK NODE  */
/*                                                               */
/*                                                               */
/*                                                               */
/*****************************************************************/

/*nodes where ASBC was skipped*/
DATA Nodes_asbc_skip;
      set Nodes_asbc;
      If ASBC_CGD ^= . THEN DELETE;
      _tail_=asbc_node;
      AFSC=asbc_afsc;
      SOC=asbc_soc;
      /*create a unique name for the arc*/
      _name_=compress(trim(ASBC_soc)||'_'||trim(ASBC_afsc)||'_'||trim(A
SBC_Course)||'_SINK');
      format _name_ $300.;
RUN;
```

```
/*nodes where asbc was attended but IFS was skipped*/
DATA Nodes_ifs_skip;
      set Nodes_ifs;
      If IFS_CGD ^= . THEN DELETE;
      _tail_=ifs_node;
      AFSC=ifs_afsc;
      SOC=ifs_soc;
      /*create a unique name for the arc*/
      _name_=compress(trim(IFS_soc)||'_'||trim(IFS_afsc)||'_'||trim(IFS
_Course)||'_SINK');
      format _name_ $300.;
RUN;

/*nodes where asbc and IFS were attended but UPT was skipped*/
DATA Nodes_upt_skip;
      set Nodes_upt;
      If UPT_CGD ^= . THEN DELETE;
      _tail_=upt_node;
      AFSC=upt_afsc;
      SOC=upt_soc;
      /*create a unique name for the arc*/
      _name_=compress(trim(UPT_soc)||'_'||trim(UPT_afsc)||'_'||trim(UPT
_Course)||'_SINK');
      format _name_ $300.;
RUN;


/*nodes where asbc was attended but IST was skipped*/
DATA Nodes_ist_skip;
      set Nodes_ist;
      If IST_CGD ^= . THEN DELETE;
      _tail_=ist_node;
      AFSC=ist_afsc;
      SOC=ist_soc;
      /*create a unique name for the arc*/
      _name_=compress(trim(IST_soc)||'_'||trim(IST_afsc)||'_'||trim(IST
_Course)||'_SINK');
      format _name_ $300.;
RUN;

/*actual attenders of UPT*/
DATA Nodes_UPT_noskip;
      set Nodes_upt;
      IF UPT_CGD=. THEN DELETE;
      _tail_=upt_node;
      AFSC=upt_afsc;
      SOC=upt_soc;
      /*create a unique name for the arc*/
      _name_=compress(trim(UPT_soc)||'_'||trim(UPT_afsc)||'_'||trim(UPT
_Course)||'_SINK');
      format _name_ $300.;
RUN;

/*actual attenders of IST*/
DATA Nodes_IST_noskip;
      set Nodes_IST;
      IF IST_CGD=. THEN DELETE;
      _tail_=ist_node;
      AFSC=ist_afsc;
```

```
      SOC=ist_soc;
      /*create a unique name for the arc*/
      _name_=compress(trim(IST_soc)||'_'||trim(IST_afsc)||'_'||trim(IST
_Course)||'_SINK');
      format _name_ $char300.;
RUN;

DATA Arcsff_final_to_sink;
      format _name_ $char300.
             _head_ $char300.
             _tail_ $char300.;


      set   Nodes_asbc_skip
            Nodes_ifs_skip
            Nodes_upt_skip
            Nodes_ist_skip
            Nodes_UPT_noskip
            Nodes_IST_noskip;

      _head_='sink';
      _cost_=0;
      _capac_=4000;
      _lo_=0;

RUN;

/*****************************************************************/
/*                                                               */
/*    CREATE AGGREGATE LIST OF ARCS FOR NETFLOW COMMAND          */
/*                ('CLOSE' ARCS NOT INCLUDED)                    */
/*                                                               */
/*                                                               */
/*****************************************************************/

/*format arcs from the source to ASBC*/
DATA ArcsFF_source_to_asbc;
      set Arcs_source_to_asbc (drop=_supdem_);

      /*take out close variables*/
      IF close=1 THEN DELETE;

      /*label columns for netflow*/
      _tail_=_node_;
      _head_=asbc_node;
      _cost_=_cost_;
      _capac_=max(maxsoc,maxafsc);
      _lo_=min(minsoc,minafsc);
      _name_=_name_;
      SOC=source_soc;
      AFSC=source_afsc;

RUN;

/*format arcs from ASBC to IFS (pilots)*/
DATA ArcsFF_asbc_to_ifs;
      set Arcs_asbc_to_ifs;

      /*take out close variables*/
```

```
      IF close=1 THEN DELETE;

      /*label columns for netflow*/
      _tail_=asbc_node;
      _head_=ifs_node;
      _cost_=_cost_;
      _capac_=maxsoc;
      _lo_=minsoc;
      _name_=_name_;
      SOC=asbc_soc;
      AFSC=asbc_afsc;

RUN;

/*format arcs from ASBC to IST (non-pilots)*/
DATA ArcsFF_asbc_to_ist;
      set Arcs_asbc_to_ist;

      /*take out close variables*/
      IF close=1 THEN DELETE;

      /*label columns for netflow*/
      _tail_=asbc_node;
      _head_=ist_node;
      _cost_=_cost_;
      _capac_=maxsoc;
      _lo_=minsoc;
      _name_=_name_;
      SOC=asbc_soc;
      AFSC=asbc_afsc;

RUN;

/*format arcs from IFS to UPT (non-pilots)*/
DATA ArcsFF_IFS_to_UPT;
      set Arcs_IFS_to_UPT;

      /*take out close variables*/
      IF close=1 THEN DELETE;

      /*label columns for netflow*/
      _tail_=IFS_node;
      _head_=UPT_node;
      _cost_=_cost_;
      _capac_=maxsoc;
      _lo_=minsoc;
      _name_=_name_;
      SOC=ifs_soc;
      AFSC=ifs_afsc;

RUN;




/*************************************************************/
/*                                                           */
```

```
/*    JOIN ALL ARCS INTO A SINGLE LIST FOR THE NETFLOW COMMAND    */
/*                                                                */
/*                                                                */
/*                                                                */
/******************************************************************/

DATA Arcsff_all;
      format _name_ $char500.
               USAFA_Leave 5.
               ROTC_Delay 5.;

      set Arcsff_asbc_to_ist
            Arcsff_asbc_to_ifs
            Arcsff_ifs_to_upt
            Arcsff_source_to_asbc
            Arcsff_final_to_sink;
RUN;


/******************************************************************/
/*                                                                */
/*         CREATE _ARC_ FILE WITH ONLY NECESSARY INFO             */
/*                  FOR THE NETFLOW COMMAND                       */
/*                                                                */
/*                                                                */
/******************************************************************/


DATA _ARC_ (Keep=_name_ _lo_ _capac_ _cost_ _tail_ _head_ USAFA_Leave
ROTC_Delay);
      format _name_ $char500.
               _tail_ $char500.
               _head_ $char500.;
      set Arcsff_all;
      if _cost_>10000 then _cost_=20000;
Run;

/******************************************************************/
/*                                                                */
/*                  CREATE SIDE CONSTRAINTS FOR ASBC                */
/*                                                                */
/*                                                                */
/*                                                                */
/******************************************************************/

/*create lists of ASBC constraints (min and max and SOC and AFSC)*/
PROC SORT DATA=ASBC; BY ASBC_Course; RUN;
PROC SORT DATA=SOC; BY SOC; RUN;
PROC SORT DATA=AFSC; BY AFSC; RUN;

PROC SQL;
            CREATE TABLE CONSTS_ASBC_SOC_MIN AS
            SELECT  A.ASBC_COURSE,
                        A.ASBC_Seats,
                        B.SOC,
                        B.minpercentasbcbysource
                  FROM ASBC AS A, SOC AS B;
QUIT;
```

```
PROC SQL;
            CREATE TABLE CONSTS_ASBC_AFSC_MIN AS
            SELECT  A.ASBC_COURSE,
                        A.ASBC_Seats,
                        C.AFSC,
                        C.minpercentasbcbyafsc
                  FROM ASBC AS A, AFSC AS C;
QUIT;

PROC SQL;
            CREATE TABLE CONSTS_ASBC_SOC_MAX AS
            SELECT  A.ASBC_COURSE,
                        A.ASBC_Seats,
                        B.SOC,
                        B.maxpercentasbcbysource
                  FROM ASBC AS A, SOC AS B;
QUIT;

PROC SQL;
            CREATE TABLE CONSTS_ASBC_AFSC_MAX AS
            SELECT  A.ASBC_COURSE,
                        A.ASBC_Seats,
                        C.AFSC,
                        C.maxpercentasbcbyafsc
                  FROM ASBC AS A, AFSC AS C;
QUIT;

/*format the constraint lists*/
DATA CONSTS_ASBC_SOC_MIN;
      set CONSTS_ASBC_SOC_MIN;
      _ROW_=trim(SOC)||'_'||trim(ASBC_COURSE)||'_min';
      course=asbc_course;
RUN;
DATA CONSTS_ASBC_SOC_MAX;
      set CONSTS_ASBC_SOC_MAX;
      _ROW_=trim(SOC)||'_'||trim(ASBC_COURSE)||'_max';
      course=asbc_course;
RUN;
DATA CONSTS_ASBC_AFSC_MIN;
      set CONSTS_ASBC_AFSC_MIN;
      _ROW_=trim(AFSC)||'_'||trim(ASBC_COURSE)||'_min';
      course=asbc_course;
RUN;
DATA CONSTS_ASBC_AFSC_MAX;
      set CONSTS_ASBC_AFSC_MAX;
      _ROW_=trim(AFSC)||'_'||trim(ASBC_COURSE)||'_max';
      course=asbc_course;
RUN;

/*combine constraints with the variables*/
/*min soc*/
PROC SORT DATA=CONSTS_ASBC_SOC_MIN; BY _ROW_; RUN;
PROC SORT DATA=Arcsff_source_to_asbc; BY _NAME_; RUN;

PROC SQL;
            CREATE TABLE Coef_ASBC_SOC_MIN AS
            SELECT  A._ROW_,
                        A.course,
                        A.SOC,
```

```
                    B._name_,
                    B.asbc_soc,
                    B.asbc_course
                FROM CONSTS_ASBC_SOC_MIN AS A, Arcsff_source_to_asbc
AS B;
QUIT;

DATA Coef_ASBC_SOC_MIN(keep=_row_ _column_ _coef_);
      set Coef_ASBC_SOC_MIN;
      IF course ^= asbc_course THEN delete;
      IF soc ^= asbc_soc THEN delete;

      _column_=_name_;

      _coef_=1;

RUN;

/*max soc*/
PROC SORT DATA=CONSTS_ASBC_SOC_MAX; BY _ROW_; RUN;
PROC SORT DATA=Arcsff_source_to_asbc; BY _NAME_; RUN;

PROC SQL;
            CREATE TABLE Coef_ASBC_SOC_MAX AS
            SELECT  A._ROW_,
                    A.course,
                    A.SOC,
                    B._name_,
                    B.asbc_soc,
                    B.asbc_course
                FROM CONSTS_ASBC_SOC_MAX AS A, Arcsff_source_to_asbc
AS B;
QUIT;

DATA Coef_ASBC_SOC_MAX(keep=_row_ _column_ _coef_);
      set Coef_ASBC_SOC_MAX;
      IF course ^= asbc_course THEN delete;
      IF soc ^= asbc_soc then delete;

      _column_=_name_;

      _coef_=1;

RUN;

/*min afsc*/
PROC SORT DATA=CONSTS_ASBC_AFSC_MIN; BY _ROW_; RUN;
PROC SORT DATA=Arcsff_source_to_asbc; BY _NAME_; RUN;

PROC SQL;
            CREATE TABLE Coef_ASBC_AFSC_MIN AS
            SELECT  A._ROW_,
                    A.course,
                    A.afsc,
                    B._name_,
                    B.asbc_afsc,
                    B.asbc_course
                FROM CONSTS_ASBC_AFSC_MIN AS A, Arcsff_source_to_asbc
AS B;
```

```
QUIT;

DATA Coef_ASBC_AFSC_MIN(keep=_row_ _column_ _coef_);
      set Coef_ASBC_AFSC_MIN;
      IF course ^= asbc_course THEN delete;
      IF afsc ^= asbc_afsc then delete;

      _column_=_name_;

      _coef_=1;

RUN;

/*max afsc*/
PROC SORT DATA=CONSTS_ASBC_AFSC_MAX; BY _ROW_; RUN;
PROC SORT DATA=Arcsff_source_to_asbc; BY _NAME_; RUN;

PROC SQL;
            CREATE TABLE Coef_ASBC_AFSC_MAX AS
            SELECT  A._ROW_,
                        A.course,
                        A.afsc,
                        B._name_,
                        B.asbc_afsc,
                        B.asbc_course
                  FROM CONSTS_ASBC_AFSC_MAX AS A, Arcsff_source_to_asbc
AS B;
QUIT;

DATA Coef_ASBC_AFSC_MAX(keep=_row_ _column_ _coef_);
      set Coef_ASBC_AFSC_MAX;
      IF course ^= asbc_course THEN delete;
      IF afsc ^= asbc_afsc then delete;

      _column_=_name_;

      _coef_=1;

RUN;

/*create RHS for ASBC constraints*/
DATA coef_asbc_afsc_max_rhs(keep=_row_ _column_ _type_ _coef_);
      set consts_asbc_afsc_max;
      _column_='_rhs_';
      _type_='LE';
      _coef_=ceil(asbc_seats*maxpercentasbcbyafsc/100);
RUN;

DATA coef_asbc_afsc_min_rhs(keep=_row_ _column_ _type_ _coef_);
      set consts_asbc_afsc_min;
      _column_='_rhs_';
      _type_='GE';
      _coef_=floor(asbc_seats*minpercentasbcbyafsc/100);
RUN;

/*create RHS for SOC constraints*/
DATA coef_asbc_soc_max_rhs(keep=_row_ _column_ _type_ _coef_);
      set consts_asbc_soc_max;
      _column_='_rhs_';
```

```
      _type_='LE';
      _coef_=ceil(asbc_seats*maxpercentasbcbysource/100);
RUN;

DATA coef_asbc_soc_min_rhs(keep=_row_ _column_ _type_ _coef_);
      set consts_asbc_soc_min;
      _column_='_rhs_';
      _type_='GE';
      _coef_=floor(asbc_seats*minpercentasbcbysource/100);
RUN;

/*constraints for ASBC class size not taking soc or afsc into account*/
/*create constraint list*/
Data Consts_asbc_max;
      set ASBC;
      course=asbc_course;
      _ROW_=trim(asbc_COURSE)||'_max';
RUN;

/*combine constraints with the variables*/
PROC SORT DATA=Consts_asbc_max; BY _ROW_; RUN;
PROC SORT DATA=Arcsff_source_to_asbc; BY _NAME_; RUN;

PROC SQL;
            CREATE TABLE Coef_ASBC_MAX AS
            SELECT  A._ROW_,
                        A.course,
                        B._name_,
                        B.asbc_course
                  FROM Consts_asbc_max AS A, Arcsff_source_to_asbc AS
B;
QUIT;

DATA Coef_ASBC_MAX(keep=_row_ _column_ _coef_);
      set Coef_ASBC_MAX;
      IF course ^= asbc_course THEN delete;

      _column_=_name_;

      _coef_=1;

RUN;

/*create RHS for ASBC MAX constraints*/
DATA Coef_ASBC_MAX_rhs(keep=_row_ _column_ _type_ _coef_);
      set Consts_asbc_max;
      _column_='_rhs_';
      _type_='LE';
      _coef_=asbc_seats;
RUN;


/*combine all asbc coef lists*/
DATA coef_asbc;
      format _type_ $8.;
      set   Coef_asbc_afsc_max
            Coef_asbc_afsc_min
            Coef_asbc_soc_max
            Coef_asbc_soc_min
```

```
            Coef_asbc_max
            Coef_asbc_afsc_max_rhs
            Coef_asbc_afsc_min_rhs
            Coef_asbc_max_rhs
            Coef_asbc_soc_max_rhs
            Coef_asbc_soc_min_rhs;
RUN;
/*done creating ASBC constrinats*/

/*****************************************************************/
/*                                                               */
/*                  CREATE SIDE CONSTRAINTS FOR IFS                */
/*                                                               */
/*                                                               */
/*                                                               */
/*****************************************************************/

/*create lists of IFS constraints (min and max by SOC)*/
PROC SORT DATA=IFS; BY IFS_Course; RUN;
PROC SORT DATA=SOC; BY SOC; RUN;

PROC SQL;
            CREATE TABLE CONSTS_IFS_SOC_MIN AS
            SELECT  A.IFS_COURSE,
                        A.IFS_Seats,
                        B.SOC,
                        B.minpercentifsbysource
                  FROM IFS AS A, SOC AS B;
QUIT;

PROC SQL;
            CREATE TABLE CONSTS_IFS_SOC_MAX AS
            SELECT  A.IFS_COURSE,
                        A.IFS_Seats,
                        B.SOC,
                        B.maxpercentifsbysource
                  FROM IFS AS A, SOC AS B;
QUIT;

/*format the constraint lists*/
DATA CONSTS_IFS_SOC_MIN;
      set CONSTS_IFS_SOC_MIN;
      _ROW_=trim(SOC)||'_'||trim(IFS_COURSE)||'_min';
      course=ifs_course;
RUN;
DATA CONSTS_IFS_SOC_MAX;
      set CONSTS_IFS_SOC_MAX;
      _ROW_=trim(SOC)||'_'||trim(IFS_COURSE)||'_max';
      course=ifs_course;
RUN;

/*combine constraints with the variables*/
/*min soc*/
PROC SORT DATA=CONSTS_IFS_SOC_MIN; BY _ROW_; RUN;
PROC SORT DATA=Arcsff_asbc_to_ifs; BY _NAME_; RUN;

PROC SQL;
            CREATE TABLE Coef_IFS_SOC_MIN AS
            SELECT  A._ROW_,
```

```
                        A.course,
                        A.SOC,
                        B._name_,
                        B.ifs_soc,
                        B.ifs_course
                    FROM CONSTS_IFS_SOC_MIN AS A, Arcsff_asbc_to_ifs AS
B;
QUIT;

DATA Coef_IFS_SOC_MIN(keep=_row_ _column_ _coef_);
      set Coef_IFS_SOC_MIN;

      IF course ^= IFS_course THEN delete;
      IF soc ^= IFS_soc THEN delete;

      _column_=_name_;

      _coef_=1;

RUN;

/*max soc*/
PROC SORT DATA=CONSTS_IFS_SOC_MAX; BY _ROW_; RUN;
PROC SORT DATA=Arcsff_asbc_to_ifs; BY _NAME_; RUN;

PROC SQL;
            CREATE TABLE Coef_IFS_SOC_MAX AS
            SELECT  A._ROW_,
                        A.course,
                        A.SOC,
                        B._name_,
                        B.ifs_soc,
                        B.ifs_course
                    FROM CONSTS_ifs_SOC_MAX AS A, Arcsff_asbc_to_ifs AS
B;
QUIT;

DATA Coef_IFS_SOC_MAX(keep=_row_ _column_ _coef_);
      set Coef_IFS_SOC_MAX;
      IF course ^= ifs_course THEN delete;
      IF soc ^= ifs_soc then delete;

      _column_=_name_;

      _coef_=1;

RUN;

/*create RHS for SOC constraints*/
DATA coef_ifs_soc_max_rhs(keep=_row_ _column_ _type_ _coef_);
      set consts_ifs_soc_max;
      _column_='_rhs_';
      _type_='LE';
      _coef_=ceil(ifs_seats*maxpercentifsbysource/100);
RUN;

DATA coef_ifs_soc_min_rhs(keep=_row_ _column_ _type_ _coef_);
      set consts_ifs_soc_min;
      _column_='_rhs_';
```

```
      _type_='GE';
      _coef_=floor(ifs_seats*minpercentifsbysource/100);
RUN;

/*create constraints for actual class size (not considering soc)*/
Data Consts_ifs_max;
      set IFS;
      course=ifs_course;
      _ROW_=trim(ifs_COURSE)||'_max';
RUN;

/*combine constraints with the variables*/
PROC SORT DATA=Consts_ifs_max; BY _ROW_; RUN;
PROC SORT DATA=Arcsff_asbc_to_ifs; BY _NAME_; RUN;

PROC SQL;
            CREATE TABLE Coef_IFS_MAX AS
            SELECT  A._ROW_,
                        A.course,
                        B._name_,
                        B.ifs_course
                  FROM Consts_ifs_max AS A, Arcsff_asbc_to_ifs AS B;
QUIT;

DATA Coef_IFS_MAX(keep=_row_ _column_ _coef_);
      set Coef_IFS_MAX;
      IF course ^= ifs_course THEN delete;

      _column_=_name_;

      _coef_=1;

RUN;

/*create RHS for IFS MAX constraints*/
DATA Coef_IFS_MAX_rhs(keep=_row_ _column_ _type_ _coef_);
      set Consts_IFS_max;
      _column_='_rhs_';
      _type_='LE';
      _coef_=ifs_seats;
RUN;

/*combine all ifs coef lists*/
DATA coef_ifs;
      set   Coef_ifs_soc_max
            Coef_ifs_soc_min
            Coef_ifs_max
            Coef_ifs_max_rhs
            Coef_ifs_soc_max_rhs
            Coef_ifs_soc_min_rhs;
RUN;
/*done creating IFS constraints*/

/*****************************************************************/
/*                                                               */
/*                  CREATE SIDE CONSTRAINTS FOR IST                */
/*                                                               */
/*                                                               */
/*                                                               */
```

```
/*****************************************************************/

/*create lists of IST constraints (min and max by SOC)*/
PROC SORT DATA=IST_nonpilot; BY IST_Course; RUN;
PROC SORT DATA=SOC; BY SOC; RUN;

PROC SQL;
            CREATE TABLE CONSTS_IST_SOC_MIN AS
            SELECT  A.IST_COURSE,
                        A.IST_Seats,
                        B.SOC,
                        B.minpercentistbysource
                  FROM IST_nonpilot AS A, SOC AS B;
QUIT;

PROC SQL;
            CREATE TABLE CONSTS_IST_SOC_MAX AS
            SELECT  A.IST_COURSE,
                        A.IST_Seats,
                        B.SOC,
                        B.maxpercentistbysource
                  FROM IST_nonpilot AS A, SOC AS B;
QUIT;

/*format the constraint lists*/
DATA CONSTS_IST_SOC_MIN;
      set CONSTS_IST_SOC_MIN;
      _ROW_=trim(SOC)||'_'||trim(IST_COURSE)||'_min';
      course=ist_course;
RUN;
DATA CONSTS_IST_SOC_MAX;
      set CONSTS_IST_SOC_MAX;
      _ROW_=trim(SOC)||'_'||trim(IST_COURSE)||'_max';
      course=ist_course;
RUN;

/*combine constraints with the variables*/
/*min soc*/
PROC SORT DATA=CONSTS_IST_SOC_MIN; BY _ROW_; RUN;
PROC SORT DATA=Arcsff_asbc_to_ist; BY _NAME_; RUN;

PROC SQL;
            CREATE TABLE Coef_IST_SOC_MIN AS
            SELECT  A._ROW_,
                        A.course,
                        A.SOC,
                        B._name_,
                        B.ist_soc,
                        B.ist_course
                  FROM CONSTS_IST_SOC_MIN AS A, Arcsff_asbc_to_ist AS
B;
QUIT;

DATA Coef_IST_SOC_MIN(keep=_row_ _column_ _coef_);
      set Coef_IST_SOC_MIN;

      IF course ^= IST_course THEN delete;
      IF soc ^= IST_soc THEN delete;
```

```
      _column_=_name_;

      _coef_=1;

RUN;

/*max soc*/
PROC SORT DATA=CONSTS_IST_SOC_MAX; BY _ROW_; RUN;
PROC SORT DATA=Arcsff_asbc_to_ist; BY _NAME_; RUN;

PROC SQL;
            CREATE TABLE Coef_IST_SOC_MAX AS
            SELECT  A._ROW_,
                        A.course,
                        A.SOC,
                        B._name_,
                        B.ist_soc,
                        B.ist_course
                  FROM CONSTS_ist_SOC_MAX AS A, Arcsff_asbc_to_ist AS
B;
QUIT;

DATA Coef_IST_SOC_MAX(keep=_row_ _column_ _coef_);
      set Coef_IST_SOC_MAX;
      IF course ^= ist_course THEN delete;
      IF soc ^= ist_soc then delete;

      _column_=_name_;

      _coef_=1;

RUN;

/*create RHS for SOC constraints*/
DATA coef_ist_soc_max_rhs(keep=_row_ _column_ _type_ _coef_);
      set consts_ist_soc_max;
      _column_='_rhs_';
      _type_='LE';
      _coef_=ceil(ist_seats*maxpercentistbysource/100);
RUN;

DATA coef_ist_soc_min_rhs(keep=_row_ _column_ _type_ _coef_);
      set consts_ist_soc_min;
      _column_='_rhs_';
      _type_='GE';
      _coef_=floor(ist_seats*minpercentistbysource/100);
RUN;

/*create constraints for actual class size (not considering soc)*/
Data Consts_ist_max;
      set IST_nonpilot;
      course=ist_course;
      _ROW_=trim(ist_COURSE)||'_max';
RUN;

/*combine constraints with the variables*/
PROC SORT DATA=Consts_ist_max; BY _ROW_; RUN;
PROC SORT DATA=Arcsff_asbc_to_ist; BY _NAME_; RUN;
```

```
PROC SQL;
            CREATE TABLE Coef_IST_MAX AS
            SELECT  A._ROW_,
                        A.course,
                        B._name_,
                        B.ist_course
                  FROM Consts_ist_max AS A, Arcsff_asbc_to_ist AS B;
QUIT;

DATA Coef_IST_MAX(keep=_row_ _column_ _coef_);
      set Coef_IST_MAX;
      IF course ^= ist_course THEN delete;

      _column_=_name_;

      _coef_=1;

RUN;

/*create RHS for IST MAX constraints*/
DATA Coef_IST_MAX_rhs(keep=_row_ _column_ _type_ _coef_);
      set Consts_IST_max;
      _column_='_rhs_';
      _type_='LE';
      _coef_=ist_seats;
RUN;

/*combine all ist coef lists*/
DATA coef_ist;
      set   Coef_ist_soc_max
            Coef_ist_soc_min
            Coef_ist_max
            Coef_ist_max_rhs
            Coef_ist_soc_max_rhs
            Coef_ist_soc_min_rhs;
RUN;
/*done creating IST constraints*/


/*****************************************************************/
/*                                                               */
/*                  CREATE SIDE CONSTRAINTS FOR UPT                */
/*                                                               */
/*                                                               */
/*                                                               */
/*****************************************************************/

/*create lists of UPT constraints (min and max by SOC)*/
PROC SORT DATA=UPT; BY UPT_Course; RUN;
PROC SORT DATA=SOC; BY SOC; RUN;

PROC SQL;
            CREATE TABLE CONSTS_UPT_SOC_MIN AS
            SELECT  A.UPT_COURSE,
                        A.UPT_Seats,
                        B.SOC,
                        B.minpercentuptbysource
                  FROM UPT AS A, SOC AS B;
QUIT;
```

```
PROC SQL;
            CREATE TABLE CONSTS_UPT_SOC_MAX AS
            SELECT  A.UPT_COURSE,
                        A.UPT_Seats,
                        B.SOC,
                        B.maxpercentuptbysource
                  FROM UPT AS A, SOC AS B;
QUIT;

/*format the constraint lists*/
DATA CONSTS_UPT_SOC_MIN;
      set CONSTS_UPT_SOC_MIN;
      _ROW_=trim(SOC)||'_'||trim(UPT_COURSE)||'_min';
      course=upt_course;
RUN;
DATA CONSTS_UPT_SOC_MAX;
      set CONSTS_UPT_SOC_MAX;
      _ROW_=trim(SOC)||'_'||trim(UPT_COURSE)||'_max';
      course=upt_course;
RUN;

/*combine constraints with the variables*/
/*min soc*/
PROC SORT DATA=CONSTS_UPT_SOC_MIN; BY _ROW_; RUN;
PROC SORT DATA=Arcsff_ifs_to_upt; BY _NAME_; RUN;

PROC SQL;
            CREATE TABLE Coef_UPT_SOC_MIN AS
            SELECT  A._ROW_,
                        A.course,
                        A.SOC,
                        B._name_,
                        B.upt_soc,
                        B.upt_course
                  FROM CONSTS_UPT_SOC_MIN AS A, Arcsff_ifs_to_upt AS B;
QUIT;

DATA Coef_UPT_SOC_MIN(keep=_row_ _column_ _coef_);
      set Coef_UPT_SOC_MIN;

      IF course ^= UPT_course THEN delete;
      IF soc ^= UPT_soc THEN delete;

      _column_=_name_;

      _coef_=1;

RUN;

/*max soc*/
PROC SORT DATA=CONSTS_UPT_SOC_MAX; BY _ROW_; RUN;
PROC SORT DATA=Arcsff_ifs_to_upt; BY _NAME_; RUN;

PROC SQL;
            CREATE TABLE Coef_UPT_SOC_MAX AS
            SELECT  A._ROW_,
                        A.course,
                        A.SOC,
```

```
                        B._name_,
                        B.upt_soc,
                        B.upt_course
                    FROM CONSTS_upt_SOC_MAX AS A, Arcsff_ifs_to_upt AS B;
QUIT;

DATA Coef_UPT_SOC_MAX(keep=_row_ _column_ _coef_);
      set Coef_UPT_SOC_MAX;
      IF course ^= upt_course THEN delete;
      IF soc ^= upt_soc then delete;

      _column_=_name_;

      _coef_=1;

RUN;

/*create RHS for SOC constraints*/
DATA coef_upt_soc_max_rhs(keep=_row_ _column_ _coef_);
      set consts_upt_soc_max;
      _column_='_rhs_';
      _coef_=ceil(upt_seats*maxpercentuptbysource/100);
RUN;

DATA coef_upt_soc_min_rhs(keep=_row_ _column_ _coef_);
      set consts_upt_soc_min;
      _column_='_rhs_';
      _coef_=floor(upt_seats*minpercentuptbysource/100);
RUN;

/*create type for SOC constraints(LE not needed because it is LE by
default)*/
DATA coef_upt_soc_min_type(keep=_row_ _column_ _coef_);
      set consts_upt_soc_min;
      _column_='_type_';
      _coef_=1;
RUN;

/*create constraints for actual class size (not considering soc)*/
Data Consts_upt_max;
      set UPT;
      course=upt_course;
      _ROW_=trim(upt_COURSE)||'_max';
RUN;

/*combine constraints with the variables*/
PROC SORT DATA=Consts_upt_max; BY _ROW_; RUN;
PROC SORT DATA=Arcsff_ifs_to_upt; BY _NAME_; RUN;

PROC SQL;
            CREATE TABLE Coef_UPT_MAX AS
            SELECT  A._ROW_,
                        A.course,
                        B._name_,
                        B.upt_course
                    FROM Consts_upt_max AS A, Arcsff_ifs_to_upt AS B;
QUIT;

DATA Coef_UPT_MAX(keep=_row_ _column_ _coef_);
```

```
      set Coef_UPT_MAX;
      IF course ^= upt_course THEN delete;

      _column_=_name_;

      _coef_=1;

RUN;

/*create RHS for UPT MAX constraints*/
DATA Coef_UPT_MAX_rhs(keep=_row_ _column_ _coef_);
      set Consts_UPT_max;
      _column_='_rhs_';
      _coef_=upt_seats;
RUN;

/*combine all upt coef lists*/
DATA coef_upt;
      set   Coef_upt_soc_max
            Coef_upt_soc_min
            Coef_upt_max
            Coef_upt_max_rhs
            Coef_upt_soc_max_rhs
            Coef_upt_soc_min_rhs
            Coef_upt_soc_min_type;
RUN;
/*done creating UPT constraints*/

/*combine all constraint files together, feed into netflow*/
DATA _constraints_;
      format _column_ $char500.;
      format _row_ $char500.;
      set   coef_ist
            coef_upt
            coef_ifs
            coef_asbc;
      format _coef_ 10.0;
RUN;

/*****************************************************************/
/*                                                               */
/*                           RUN THE NETFLOW COMMAND
*/
/*                                                               */
/*                                                               */
/*                                                               */
/*****************************************************************/

proc netflow
      nodedata=_node_
      arcdata=_arc_
      condata=_constraints_ sparsecondata
      conout=_basesolution_
      MAXIT1= 99999
      sink='sink';
      RESET
      zero2=0.0001;
run;
```

```
/*format the output from the names of the variables*/
data _basesolution_;
      format end_class $90.;
      set _basesolution_;
      roundflow=round(_flow_,1);
      SOC=scan(_tail_,1,'_');
      AFSC=scan(_tail_,2,'_');
      start_class=scan(_tail_,3,'_');
      if _head_='sink' then end_class='sink';
      else end_class=scan(_head_,3,'_');
      SOC=scan(_tail_,1,'_');
      AFSC=scan(_tail_,2,'_');
      roundcost=round(_cost_,1);
      roundfcost=roundflow*roundcost;
RUN;

/*run netflow again on costs squared to see if the output changes to be
more uniform*/

data _squarearc_;
      set _arc_;
      _squarecost_=_cost_*_cost_/100; /*divide by a hundered to bring
down machine round off error*/
      _cost_=_squarecost_;
RUN;

proc netflow
      nodedata=_node_
      arcdata=_squarearc_
      condata=_constraints_ sparsecondata
      conout=_squaredsolution_
      MAXIT1= 99999
      sink='sink';
      RESET
      zero2=0.0001;
run;

/*format the output from the names of the variables*/
data _squaredsolution_;
      format end_class $90.;
      set _squaredsolution_;
      roundflow=round(_flow_,1);
      SOC=scan(_tail_,1,'_');
      AFSC=scan(_tail_,2,'_');
      start_class=scan(_tail_,3,'_');
      if _head_='sink' then end_class='sink';
      else end_class=scan(_head_,3,'_');
      SOC=scan(_tail_,1,'_');
      AFSC=scan(_tail_,2,'_');
      roundcost=round(_cost_,1);
      roundfcost=roundflow*roundcost;
RUN;


PROC SORT DATA=_basesolution_; BY end_class _tail_ _head_ _name_; RUN;
PROC SORT DATA=_squaredsolution_; BY end_class _tail_ _head_ _name_;
RUN;

data _basesolution_;
```

```
      set _basesolution_;
      baseflow=roundflow;
      basecost=_cost_;
run;

data _squaredsolution_;
      set _squaredsolution_;
      squaredflow=roundflow;
      squaredcost=sqrt(_cost_*100);
run;

/*create output of all data that is to be read into the excel sheet for
further processing*/
data _alloutput_(keep=start_class end_class AFSC SOC baseflow basecost
squaredflow _capac_ USAFA_Leave ROTC_Delay Course Baseskip
Squaredskip);
      merge _basesolution_
              _squaredsolution_;
    by end_class _tail_ _head_ _name_;
      format Course $char30.;
      if    index(end_class,'ASBC')=1 or end_class='SKIPASBC' then
Course='ASBC';
      else if index(end_class,'IFS')=1 or end_class='SKIPIFS' then
Course='IFS';
      else if index(end_class,'IST')=1 or end_class='SKIPIST' or
end_class='SKIPUPT' then Course='IST'||trim(AFSC);
      else if end_class='sink' then Course='sink';
      else Course='error';
      if index(end_class,'SKIP')=1 then Baseskip = baseflow;
      if index(end_class,'SKIP')=1 then Squaredskip = Squaredflow;
run;

/*create list of all arcs that are actually used (not including the
skip arcs)*/
data _basemaxfind_;
      set _basesolution_;
      if _flow_=0 or _cost_=20000 then delete;
RUN;

data _squaredmaxfind_;
      set _squaredsolution_;
      if _flow_=0 or _cost_=20000*20000/100 then delete;
RUN;

/*make lists to be put through a MAX netflow function*/

/*list of all used arcs that end at an asbc class*/
data _basemaxfindsource_(keep=_name_ _head_ _tail_ _cost_ SOC AFSC);
      set _basemaxfind_;
      if index(end_class,'ASBC')=1 then;
      ELSE DELETE;
      _head_=_tail_;
      _tail_='source';
      _name_=trim(SOC)||'_'||trim(AFSC)||'_source_'||trim(start_class);
      _cost_=0;
run;

data _squaredmaxfindsource_(keep=_name_ _head_ _tail_ _cost_ SOC AFSC);
      set _squaredmaxfind_;
```

```
      if index(end_class,'ASBC')=1 then;
      ELSE DELETE;
      _head_=_tail_;
      _tail_='source';
      _name_=trim(SOC)||'_'||trim(AFSC)||'_source_'||trim(start_class);
      _cost_=0;
run;




/*combine the source arc list with the used arc list*/
data _basemaxfindarc_ (Keep=_name_ _head_ _tail_ _cost_ SOC AFSC);
      set _basemaxfind_ _basemaxfindsource_;
RUN;

data _squaredmaxfindarc_ (Keep=_name_ _head_ _tail_ _cost_ SOC AFSC);
      set _squaredmaxfind_ _squaredmaxfindsource_;
RUN;


/*run a maxflow problem to see the highest cost single flow through the
network*/
proc netflow
      max
      arcdata=_basemaxfindarc_
      arcout=_basemaxsolution_
      MAXIT1= 99999
      sink='sink'
      source='source'
      supply=1;
      RESET
      zero2=0.0001;
run;

proc netflow
      max
      arcdata=_squaredmaxfindarc_
      arcout=_squaredmaxsolution_
      MAXIT1= 99999
      sink='sink'
      source='source'
      supply=1;
      RESET
      zero2=0.0001;
run;


/*show only the arcs that had the flow on it*/
Data _basemaxsolution_;
      set _basemaxsolution_;
      if _flow_=0 then delete;
RUN;

Data _squaredmaxsolution_;
      set _squaredmaxsolution_;
      if _flow_=0 then delete;
      _cost_=sqrt(_cost_*100);
```

```
RUN;
```

Appendix B: Output Table

Name	Min Leave	Max Delay	Max Blend	Arc Costs	Average Ineffective Days	Number Of People Skipping
1	0	0	50	Base	95.72156672	897
2	0	0	50	Squared	96.1795402	897
3	0	0	75	Base	98.78071521	771
4	0	0	75	Squared	99.16865337	771
5	0	0	100	Base	98.39266769	766
6	0	0	100	Squared	98.79677001	766
7	0	180	50	Base	68.95117001	897
8	0	180	50	Squared	69.28774997	897
9	0	180	75	Base	68.60678044	771
10	0	180	75	Squared	69.4205564	771
11	0	180	100	Base	67.74520391	766
12	0	180	100	Squared	68.70820739	766
13	0	365	50	Base	57.47385814	897
14	0	365	50	Squared	57.70984517	897
15	0	365	75	Base	53.42949812	771
16	0	365	75	Squared	54.23915051	771
17	0	365	100	Base	53.41045645	766
18	0	365	100	Squared	53.73959308	766
19	30	0	50	Base	96.46900599	918
20	30	0	50	Squared	96.64017606	918
21	30	0	75	Base	99.07380248	780
22	30	0	75	Squared	99.46256653	780
23	30	0	100	Base	98.51691619	766
24	30	0	100	Squared	98.92054698	766
25	30	180	50	Base	69.72829332	918
26	30	180	50	Squared	70.59461418	918
27	30	180	75	Base	68.66505023	780
28	30	180	75	Squared	69.6756662	780
29	30	180	100	Base	67.78554189	766
30	30	180	100	Squared	68.53426087	766
31	30	365	50	Base	61.38057267	918
32	30	365	50	Squared	60.61832698	918
33	30	365	75	Base	54.57169202	780
34	30	365	75	Squared	55.67746859	780
35	30	365	100	Base	52.717981	766
36	30	365	100	Squared	53.8926198	766
37	60	0	50	Base	96.46900599	918
38	60	0	50	Squared	96.64017606	918
39	60	0	75	Base	99.07380248	780
40	60	0	75	Squared	99.46256653	780
41	60	0	100	Base	98.51691619	766
42	60	0	100	Squared	98.92054698	766
43	60	180	50	Base	70.14718251	918
44	60	180	50	Squared	70.62019812	918
45	60	180	75	Base	68.35724138	780
46	60	180	75	Squared	69.56184364	780
47	60	180	100	Base	67.76217831	766
48	60	180	100	Squared	68.9982491	766
49	60	365	50	Base	61.93429233	918
50	60	365	50	Squared	60.36655987	918
51	60	365	75	Base	54.47318757	780
52	60	365	75	Squared	56.08574808	780
53	60	365	100	Base	52.69385654	766
54	60	365	100	Squared	54.48980314	766

Name	Average Post USAFA Leave	Min Post USAFA Leave	Average Post ROTC Delay	Max Post ROTC Delay	Length of Worst Schedule
1	59.1200318	33	0	0	994
2	56.67329094	33	0	0	558
3	59.31319555	33	0	0	1029
4	59.31319555	33	0	0	526
5	59.46343402	33	0	0	1060
6	59.52782194	33	0	0	537
7	54.31240064	33	141.8213966	180	981
8	54.93481717	33	145.9104745	180	542
9	59.31319555	33	141.8213966	180	956
10	59.16295707	33	147.3639212	180	540
11	59.76391097	33	139.8795882	180	956
12	58.92686804	33	147.4167413	180	501
13	53.58267091	33	232.4668756	365	692
14	54.24801272	33	231.8146822	365	542
15	57.33863275	33	222.3222919	365	607
16	57.29570747	33	235.5326768	365	540
17	56.63036566	33	251.743957	365	642
18	57.59618442	33	247.1750224	365	542
19	60	60	0	0	875
20	60	60	0	0	558
21	60	60	0	0	960
22	60	60	0	0	526
23	60	60	0	0	1021
24	60	60	0	0	526
25	60	60	137.980752	180	751
26	60	60	143.0062668	180	542
27	60	60	140.6347359	180	956
28	60	60	147.02641	180	542
29	60	60	138.905103	180	981
30	60	60	143.4292748	180	498
31	60	60	239.5425246	365	692
32	60	60	224.5259624	365	542
33	60	60	223.7121755	365	697
34	60	60	247.3039391	365	540
35	60	60	219.6956132	365	746
36	60	60	244.4171889	365	498
37	60	60	0	0	922
38	60	60	0	0	558
39	60	60	0	0	1057
40	60	60	0	0	526
41	60	60	0	0	985
42	60	60	0	0	521
43	60	60	142.1714414	180	722
44	60	60	143.0237243	180	542
45	60	60	140.0040286	180	956
46	60	60	146.0031334	180	540
47	60	60	138.2538048	180	897
48	60	60	147.0290958	180	508
49	60	60	243.4471799	365	692
50	60	60	228.0326768	365	542
51	60	60	221.606983	365	711
52	60	60	250.0519248	365	540
53	60	60	220.8679499	365	567
54	60	60	250.7802149	365	542

Bibliography

AETC. *AETC significant events.* Retrieved August 7, 2007, http://www.aetc.af.mil/library/history/aetcsignificantevents/1990-99.asp 1999.

Alcock, Shawn. *USAF initial flight screening program.* Retrieved May 21, 2007. http://www.baseops.net/militarypilot/usaf_ift.html. October 2006.

ASBC-CC. *ASBC mission statement.* Retrieved August 7, 2007, http://asbc.maxwell.af.mil/mission.htm#mission. 2006.

Bard, J., Binici, C., & deSilva, A. (2003). Staff scheduling at the united states postal service. Computers & Operations Research, 30(5), 745-771.

Bazaraa, M. S., Jarvis J. J., & Sherali, H. D. (2005). *Linear Programming and Network Flows (Third Edition).* John Wily & Sons, Inc.

Chu, H. D., Eric, G., & Ellis, L. J. (1997). *Solving large scale crew scheduling problems*. European Journal of Operational Research, 97, 260-268.

Diamond Aircraft. *USAF-IFS program launched.* Retrieved May 21, 2007, from http://www.diamondair.com/news/2007_02_09.php February 2007.

Ernst, A. T., Jiang, H., Krishnamoorthy, M., & Sier, D. (2004a). *Staff scheduling and rostering: A review of applications, methods and models*. European Journal of Operational Research, 153(1), 3-27.

Ernst, A., H. Jiang, M. Krishnamoorthy, B. Owens , & Sier, D. (2004b). *An annotated bibliography of personnel scheduling and rostering*. Annals of Operations Research, Special Issue on Staff Scheduling and Rostering, 127, 21-141.

Goodfellow. *315th Training Squadron.* Retrieved August 22, 2007, http://www.goodfellow.af.mil/library/factsheets/factsheet.asp?id=6600 1999.

Knighton, Shane. A. *An Optimal Network-Based Approach to Scheduling and Re-Rostering Continuous Heterogeneous Workforces*. Dissertation, Arizona State University. August 2005.

Love, R. R.,Jr., & Hoey, J. (1990). *Management science improves fast-food operations*. Interfaces (Providence), 20(2), 21.

Nemhauser, G. L. and Wolsey, L. A. (1997) *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc.

Oliver, Bradley. R. *Optimizing the Undergraduate Pilot Training Scheduling Process*. Graduate Research Project, Air Force Institute of Technology. March 2007.

Raghavachari, M. (1976). *Constructive method to recognize the total unimodularity of a matrix*. Zeitschrift fuer Operations Research, Serie A: Theorie, 20(1), 59-61.

Tyndall. *325th Air Control Squadron.* Retrieved August 22, 2007, http://www.tyndall.af.mil/library/factsheets/factsheet.asp?id=4863 1999.

Vazirani, V. V. (2004). *Approximation Algorithms*. Springer.

Yan, S., & Chang, J. (2002). *Airline cockpit crew scheduling*. European Journal of Operational Research, 136(3), 501-511.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No.

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
14-12-2007	Master's Thesis	Aug 2006 – Dec 2007

4. TITLE AND SUBTITLE

A NETWORK FLOW APPROACH TO THE INITIAL SKILLS TRAINING SCHEDULING PROBLEM

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Illig, Anthony A., Second Lieutenant, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Street, Building 642
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GOR/ENS/07-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AETC/SAS
Attn: Capt. Ryan F. Caulk
151 J St. East, Suite 2 DSN: 487-0872
Randolph AFB, TX 78150 e-mail: ryan.caulk@randolph.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The United States Air Force commissions new officers as they complete their undergraduate degree or their officer commissioning training. These officers are commissioned frequently throughout the calendar year, sometimes in large groups. In order to perform their job to the best of their abilities, they require proper training. With this in mind, it seems only natural that there should exist a mathematical, repeatable and measurable method for scheduling these officers into their training courses to have them fully trained and available for Air Force duties in the timeliest manner.

This thesis demonstrates that the goal of efficiently and effectively scheduling officers into their training courses is not being met and provides a method which can be utilized by the Air Force to ensure that the goal is actually achieved. The formulation is based upon a minimum cost network flow problem used to perform the scheduling. The algorithm solves the problem in a polynomial amount of time (in terms of the problem size) and gives the user measurable and comparable results. Further, numerous problem instances are presented in order to explore different decision alternatives which will enable the Air Force senior leadership to make informed decisions.

15. SUBJECT TERMS
Scheduling, Minimum Cost Network Flow, Total Unimodularity, Personnel Scheduling, Optimization

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
1. REPORT	**2. ABSTRACT**	**c. THIS PAGE**			August G. Roesener, Maj., USAF (ENS)
U	**U**	**U**	**UU**	115	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-6565, ext 4539; e-mail: august.roesener@afit.edu

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18